PUSH TECHNOLOGY

Diffusion 5.5 User Guide

Contents

List of Figures 9

List of Tables 14

Part I: Introduction 19

What's new in Diffusion 5.5? 20
What's new in Diffusion 5.1? 22
What's new in Diffusion 5.0? 23
Upgrading 25
Interoperability 25
Upgrading from version 4.x to version 5.0 26
Upgrading from version 5.0 to version 5.1 30
Upgrading from version 5.1 to version 5.5 33
Upgrading to a new patch release 39

Part II: Design Guide 40

Overview 41
Architecture 41
Data distribution 43
Diffusion server 44
Publishers 45
Topics 46
Clients 47
Diffusion APIs 48
Designing your data model 48
Topic tree 49
Topic naming 50
Topics 51
Single value topics 52
Record topics 53
DEPRECATED: Delegated topics 53
Stateless topics 54
Routing topics 55
Slave topics 55
Custom topics 56
Protocol buffer topics 56

Paged string topics ... 57
Paged record topics ... 58
Topic notify topics ... 59
Service topics ... 59
Child list topics ... 60
Topic attributes ... 60
Data format ... 60
Metadata ... 61
Byte encoding of content ... 63
Publication ... 64
Publishing data ... 65
Subscribing to topics ... 66
Distributing and viewing data as pages ... 75
Conflation ... 76
Types of message conflation ... 77
How conflation works ... 78
Messaging ... 79
Designing your solution ... 80
Servers ... 81
Replication ... 82
Session replication ... 83
Topic replication ... 86
Failover of active update sources ... 87
User-written components ... 90
Publishers ... 90
Clients ... 95
Other user-written components ... 100
Third party components ... 102
Web servers ... 102
Load balancers ... 103
Proxies ... 104
JMS ... 104
Example solutions ... 119
Example: Simple solution ... 120
Example: Fan-out distribution ... 120
Example: A solution using clients ... 122
Support ... 123
System requirements for the Diffusion server ... 123
Platform support for the Diffusion libraries ... 125
Protocols supported ... 126
Browsers supported ... 127
Browser limitations ... 127
Security ... 134
Role-based authorization ... 134
Permissions ... 138
Pre-defined roles ... 141
Authentication ... 142
User-written authentication handlers ... 145
System authentication handler ... 147
Authorization handlers ... 148

Part III: Developer Guide ... 152
Developing a publisher ... 153
Writing a publisher ... 153

Creating a Publisher class ... 153
Publisher startup ... 153
Data state ... 154
Data inputs ... 154
Handling client subscriptions ... 155
Publishing messages ... 156
Topic locking ... 157
Handling clients ... 157
Publisher closedown ... 158
Testing a publisher ... 159
Client groups ... 159
Client notifications ... 160
Adding a ClientListener ... 161
Using DefaultClientListener ... 161
Developing a client ... 162
Unified API ... 162
Key concepts in the Unified API ... 163
Features ... 184
Java Unified API ... 351
.NET Unified API ... 395
JavaScript Unified API ... 447
C Unified API ... 467
Classic API ... 497
Table of Classic APIs ... 497
Java API ... 499
.NET API ... 503
JavaScript API ... 507
ActionScript API ... 514
Silverlight API ... 519
iOS Classic API ... 523
Android Classic API ... 531
C API ... 538
diffusion-wrapper.js ... 539
Testing ... 542
Flex/Flash client ... 542
Java client test tool ... 546
JavaScript client test tool ... 550
Silverlight client test tool ... 551
Windows client test tool (.NET) ... 553
Stress test tuning ... 556
Stress test ... 557
Benchmarking suite ... 558
Test tools ... 558

Part IV: Administrator Guide ... 559

Installing ... 560
System requirements for the Diffusion server ... 560
Obtaining a Diffusion license ... 561
Installing the Diffusion server ... 562
Installing Diffusion using the headless installer ... 563
Installing Diffusion using Red Hat Package Manager ... 564
Installing Diffusion using Docker ... 564
Next steps with Docker ... 565
Updating your license file ... 566

Installed files... 567
Verifying your installation... 568
Web server installation... 570
Configuration... 572
XML configuration... 573
Programmatic configuration... 575
Using the configuration API... 576
The configuration tree... 578
Configuring replication... 579
Replication.xml... 580
Configuring your datagrid provider... 581
Configuring the JMS adapter... 583
JMSAdapter.xml... 584
Example: Configuring JMS providers for the JMS adapter... 592
Example: Configuring topics for use with the JMS adapter... 593
Example: Configuring pubsub with the JMS adapter... 594
Example: Configuring messaging with the JMS adapter... 595
Example: Configuring the JMS adapter to work with JMS services... 596
DEPRECATED: Configuring the legacy JMS adapter version 5.1... 597
Aliases.xml... 605
ConnectionValidationPolicy.xml... 606
Connectors.xml... 607
Env.xml... 613
Logs.xml... 613
Management.xml... 616
Mime.xml... 617
Publishers.xml... 618
Server.xml... 622
Statistics.xml... 637
SubscriptionValidationPolicy.xml... 640
WebServer.xml... 642
Additional XML files... 648
Server... 648
Server basics... 648
Starting the server... 649
Running from within a Java application... 650
Concurrency... 652
Connectors... 655
Load balancers... 657
Web server... 660
Interaction with publishers... 660
Security... 661
System management... 661
Going to production... 662
General management... 663
Classic deployment... 663
Hot deployment... 663
What's in a DAR file?... 664
Building a DAR file... 665
Deployment methods... 666
Using JMX... 667
Configuring the Diffusion JMX connector server... 668
Configuring a remote JMX server connector... 670
Configuring a local JMX connector server... 671
Using Java VisualVM... 671

Using JConsole....673
MBeans....676
Statistics....684
Configuring statistics....687
Diffusion monitoring console....688
Basic integration with Splunk....699
Network security....702
Tuning....704
Buffer sizing....704
Message sizing....706
Client queues....707
Client multiplexers....707
Connectors....708
Thread pools....708
Client reconnection....711
Client failover....715
Client throttling....717
Memory considerations....718
Garbage collection (Java HotSpot VM)....718
Platform-specific issues....718
Socket issues....718
Publisher design....720
Diagnostics....721
Logging....721
Logging API....725
Logging in the .NET API....726
Connection counts....727
JavaScript diagnostics....727
Flex and Flash diagnostics....731
Debugging a publisher....732
Log Messages....735
Introspector....768
Supported platforms....768
Installing from update site....768
Installing subsequent plugin updates....772
Uninstalling....772
Opening the Diffusion perspective....773
Adding servers....775
Opening servers....776
Exploring the topics....776
Getting topic values....776
Configuring columns....778
Ping servers....778
Count topics....778
Using the clients view....779
Ping....779
Statistics....780
Topics....780
Logging....780
Server logs....781
Property obfuscator....781
Demos....782
Demos....782
Building the demos using mvndar....783
Tools....784

Tools for Amazon Elastic Compute Cloud (EC2)..... 784
Tools for Joyent..... 786
Using Maven to build Java Diffusion applications..... 786
Building client applications with Maven..... 787
Building publishers and other server application code with Maven..... 788

Appendices.....790

Appendix A: Document conventions.....791

Appendix B: Glossary.....792
A.....793
C.....793
D.....795
E.....796
F.....796
G.....797
H.....797
I.....798
J.....798
L.....799
M.....800
N.....800
P.....801
Q.....803
R.....803
S.....804
T.....806
U.....807
V.....808
W.....808
X.....808

Appendix C: Trademarks.....810

Appendix D: Copyright Notices..... 812
ANTLR.....814
Apache Commons Codec.....814
Apache Portable Runtime.....814
Bootstrap..... 814
CocoaAsyncSocket..... 815
concurrent-trees.....815
CQEngine.....815
cron4j.....815
d3.....815
FastColoredTextBox..... 816
Fluent validation..... 816
Fluidbox.....816
GeoIP API..... 816
GeoLite City Database.....816

geronimo-jms_1.1_spec....816
Google code prettify....817
hashmap....817
Hazelcast....817
HPPC....818
htmlcompressor....818
inherits....818
JCIP Annotations....818
JCTools....818
jQuery....818
json-simple....819
JZlib....819
Knockout....819
loglevel....819
long....819
Metrics....819
Minimal JSON....820
NLog....820
opencsv....820
OpenSSL....820
PCRE....820
Picocontainer....821
Protocol Buffers....821
Rickshaw....821
Servlet API....822
SLF4J....822
SocketRocket....822
Tabber....822
Tapestry (Plastic)....822
TrueLicense....823
when....823
Licenses....823

List of Figures

Figure 1: Basic architectural components....41
Figure 2: A scalable, resilient architecture....42
Figure 3: Components in a Diffusion server....44
Figure 4: Topics....46
Figure 5: Example topic tree....49
Figure 6: Pub-sub model....64
Figure 7: Message flow without conflation enabled....77
Figure 8: Message flow with simple replace conflation enabled....77
Figure 9: Message flow with simple append conflation enabled....77
Figure 10: Message flow with merge and replace conflation enabled....78
Figure 11: A client registers a handler on part of the topic tree....79
Figure 12: A client can send messages through a topic path to known client sessions....80
Figure 13: Information sharing using a datagrid....83
Figure 14: Session replication....83
Figure 15: Topic replication....86
Figure 16: Failover of an active update source....88
Figure 17: Distributors....94
Figure 18: Aggregators....95
Figure 19: JMS message structure....106

Figure 20: Basic mapping from a JMS message to a Diffusion message..... 107
Figure 21: Basic mapping from a Diffusion message to a JMS message..... 107
Figure 22: Mapping from a JMS message to and from JSON in a Diffusion message..... 108
Figure 23: JMS adapter: Publishing from JMS to Diffusion..... 109
Figure 24: JMS adapter: Message flow from Diffusion to JMS..... 110
Figure 25: JMS adapter: Message flow from JMS to Diffusion..... 111
Figure 26: JMS adapter: Request-response message flow..... 112
Figure 27: Subscription flow..... 116
Figure 28: Sending flow from a Diffusion client to a JMS topic (or queue)..... 117
Figure 29: Request-reply initiated by a JMS client and serviced by a Diffusion client..... 118
Figure 30: Request-reply initiated by a Diffusion client and serviced by a JMS client..... 119
Figure 31: A simple solution..... 120
Figure 32: Fan-out using distributed topics..... 121
Figure 33: Clients for different purposes..... 122
Figure 34: Topic scope example..... 140
Figure 35: Authentication process for clients..... 143
Figure 36: A composite authentication handler..... 146
Figure 37: Session state model..... 165
Figure 38: A callback..... 170
Figure 39: A stream..... 171
Figure 40: Flow from a subscribing client to the client that handles a missing topic subscription..... 229
Figure 41: Diffusion wrapper..... 540
Figure 42: Flex client: Connection tab..... 543
Figure 43: Flex client: Send tab..... 544
Figure 44: Flex client: Messages tab..... 545

Figure 45: Flex client: Log tab....546
Figure 46: External client tester: Connection tab....547
Figure 47: External client tester: Send tab....548
Figure 48: External client tester: Messages tab....549
Figure 49: External client tester: Message details window....550
Figure 50: JavaScript test tool....551
Figure 51: Silverlight test tool: Connection tab....551
Figure 52: Silverlight test tool: Send tab....552
Figure 53: Silverlight test tool: Messages tab....553
Figure 54: Windows client test tool: Connection tab....554
Figure 55: Windows client test tool: Send tab....555
Figure 56: Windows client test tool: Messages tab....556
Figure 57: Thread diagram....655
Figure 58: Sticky-IP in F5 BIG-IP....659
Figure 59: Example folder structure inside a DAR file....664
Figure 60: Connecting to Diffusion JMX....667
Figure 61: Java VisualVM: Overview tab....672
Figure 62: JConsole New Connection dialog: Remote Process....673
Figure 63: JConsole New Connection dialog: Remote Process....674
Figure 64: JConsole New Connection dialog: Local Process....675
Figure 65: The server MBean stopController operation showing in JConsole...677
Figure 66: Logging in the monitoring console....690
Figure 67: The default console layout....691
Figure 68: The table of publishers....691
Figure 69: Publisher statistics graphs....692
Figure 70: The table of topics....692
Figure 71: Details of the topic publishing the CPU load of the host server....693

Figure 72: The table of clients..... 693
Figure 73: The table of log entries..... 695
Figure 74: Editing the Access Policy..... 696
Figure 75: Notification that the Diffusion server has stopped..... 696
Figure 76: The default Diffusion Details panel..... 697
Figure 77: Editing the properties of the Diffusion Details panel..... 697
Figure 78: Visualizing the CPU load on a server at a specific time..... 698
Figure 79: Editing and adding to the set of topics for this panel..... 698
Figure 80: Welcome tab of the Splunk web UI..... 700
Figure 81: The Splunk Set source type dialog..... 700
Figure 82: The Data Preview panel..... 701
Figure 83: The Splunk search summary panel..... 701
Figure 84: Reconnection scenario..... 714
Figure 85: Normal and throttled client queues..... 717
Figure 86: Firefox Console..... 728
Figure 87: Chrome's console..... 728
Figure 88: Internet Explorer console..... 729
Figure 89: Opera console..... 730
Figure 90: Safari's console..... 731
Figure 91: New Java Project..... 733
Figure 92: Creating a new Java class..... 734
Figure 93: Example classpath entries..... 735
Figure 94: Adding a repository..... 768
Figure 95: Install dialog..... 769
Figure 96: Accept the license agreement..... 770
Figure 97: Click OK..... 770
Figure 98: Restarting..... 771

Figure 99: About Eclipse dialog..... 772
Figure 100: Installed plugins..... 773
Figure 101: Perspective..... 774
Figure 102: Views..... 774
Figure 103: Add a server..... 775
Figure 104: Edit server details..... 776
Figure 105: View topic values..... 777
Figure 106: Re-order columns..... 778
Figure 107: Ping a server..... 778
Figure 108: Topic count..... 779
Figure 109: Ping clients..... 780
Figure 110: Server log entries..... 781
Figure 111: Property Obfuscator dialog..... 782

List of Tables

Table 1: API features removed in version 5.0 27
Table 2: API features deprecated in version 5.0 28
Table 3: API features removed in version 5.1 30
Table 4: API features deprecated in version 5.1 31
Table 5: API features removed in version 5.5 34
Table 6: API features deprecated in version 5.5 35
Table 7: Restricted characters 50
Table 8: Data types for metadata fields 62
Table 9: Types of byte encoding 63
Table 10: Encoding support transports 64
Table 11: Types of topic selector 66
Table 12: Descendant pattern qualifiers 68
Table 13: Selector examples 74
Table 14: Supported protocols by client 97
Table 15: Supported platforms and transport protocols for the client libraries 125
Table 16: Supported protocols by client 126
Table 17: Supported browsers 127
Table 18: Browser Plugins 127
Table 19: Internet Explorer support for WebSocket 128

Table 20: Firefox support for WebSocket....128
Table 21: Chrome support for WebSocket....128
Table 22: Safari support for WebSocket....129
Table 23: Opera support for WebSocket....129
Table 24: iOS support for WebSocket....129
Table 25: Android support for WebSocket....129
Table 26: Internet Explorer support for CORS....130
Table 27: Firefox support for CORS....130
Table 28: Chrome support for CORS....130
Table 29: Safari support for CORS....130
Table 30: Opera support for CORS....131
Table 31: iOS support for CORS....131
Table 32: Android support for CORS....131
Table 33: Internet Explorer maximum supported connections....132
Table 34: Firefox maximum supported connections....132
Table 35: Chrome maximum supported connections....132
Table 36: Safari maximum supported connections....132
Table 37: Opera maximum supported connections....132
Table 38: iOS maximum supported connections....133
Table 39: Android maximum supported connections....133
Table 40: List of topic-scoped permissions....139
Table 41: List of global permissions....141
Table 42: Client operations that require authentication....144
Table 43: Types of authentication handler....146
Table 44: Authorization handler methods....149
Table 45: Client listener notifications....160
Table 46: Session filter search clause operators....167

Table 47: Session filter boolean operators....168
Table 48: Data types for metadata fields....174
Table 49: Supported platforms and transport protocols for the client libraries....352
Table 50: Capabilities provided by the API....352
Table 51: Capabilities provided by the library....353
Table 52: Supported platforms and transport protocols for the client libraries....395
Table 53: Capabilities provided by the API....395
Table 54: Capabilities provided by the library....397
Table 55: Supported platforms and transport protocols for the client libraries....448
Table 56: Capabilities provided by the API....448
Table 57: Capabilities provided by the library....449
Table 58: Supported platforms and transport protocols for the client libraries....467
Table 59: Supported platforms and transport protocols for the client libraries....467
Table 60: Capabilities provided by the API....467
Table 61: Capabilities provided by the library....468
Table 62: Feature matrix....498
Table 63: Java APIs....499
Table 64: Connection types....501
Table 65: Types of connection that can be specified from the .NET client....504
Table 66: JavaScript functions called on events....508
Table 67: Supported platforms and transport protocols for the client libraries....523
Table 68: Capabilities provided by the API....524
Table 69: Capabilities provided by the library....524

Table 70: Supported platforms and transport protocols for the client libraries....531
Table 71: Capabilities provided by the API....531
Table 72: Capabilities provided by the library....532
Table 73: Tuning changes for stress testing....556
Table 74: Testing tools....558
Table 75: Installed files....567
Table 76: Tools and utilities....568
Table 77: XML Value types....573
Table 78: Properties that can be specified when configuring the JMS adapter. 599
Table 79: Connectors properties....655
Table 80: Connection restrictions....656
Table 81: Routing strategies....659
Table 82: Client security....702
Table 83: Values that can be configured for a thread pool....709
Table 84: Events that a thread pool notification handler can act on....710
Table 85: Log levels....722
Table 86: Fields included in the logs....723
Table 87: Mapping between LogWriter methods and Diffusion log levels....725
Table 88: Location of the flashlog.txt file....732
Table 89: Location of the policyfiles.txt file....732
Table 90: Client properties in the Eclipse client view....779
Table 91: Demos provided with the Diffusion server....782
Table 92: Targets....785
Table 93: Properties for targets start, stop and status....785
Table 94: Additional properties for targets deploy and undeploy....785
Table 95: Artifacts....786
Table 96: Typographic conventions used in this manual....791

Part I

Introduction

Welcome to the Push Technology User Manual for Diffusion™

Diffusion is a development software platform that removes the complexity and associated challenges of developing for scale, coping with the explosion of data across networks, delivering a rich application experience and real-time conversational interactions

Deployed in your organization's own data centers or in the cloud, Diffusion gives developers the toolkit to create high performance, value driven and reliable web and mobile applications.

Diffusion is able to intelligently understand data to automatically remove out of date or redundant data. As a result, you can use it to efficiently distribute fast-changing data to a large number of simultaneously connected users.

This offers a rich application experience for end users as they receive only relevant and up to date data. It also enables you to reduce the amount of infrastructure and bandwidth consumption required to distribute data to web and mobile applications on any internet connected device.

The manual is regularly updated, but if you require further help, see the articles and forums in our Support Center: http://support.pushtechnology.com.

In this section:

- What's new in Diffusion 5.5?
- What's new in Diffusion 5.1?
- What's new in Diffusion 5.0?
- Upgrading

What's new in Diffusion 5.5?

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

A complete list of the latest updates to Diffusion can be found in the Release Notes available at http://docs.pushtechnology.com/docs/5.5.3/release/ReleaseNotice.html.

JavaScript® Unified API

A JavaScript version of the Unified API is now available. This API contains both control and standard capabilities.

For version 5.5, the JavaScript Classic API (the API used in version 4 and earlier) is still supported.

For more information, see JavaScript Unified API.

New security model

Diffusion is now secured using a role-based model. Actions that can be performed on the server by client sessions are controlled by permissions. The client session must be assigned a role that has the required permission to be able to perform the action.

For more information, see Role-based authorization on page 134.

Console authentication

To access the Diffusion console you must now authenticate with a principal and a password. The information that you can view through the console and the actions you can take are controlled by the permissions assigned to the roles associated with that principal.

For more information, see Diffusion monitoring console on page 688.

Session properties

Information about client sessions is now available to Unified API clients using control features. These session properties can be used to filter client sessions.

For more information, see Session properties on page 165.

New JMS adapter

A new JMS adapter is available that provides enhanced functionality. The new JMS adapter provides more powerful configuration options and abstracts all JMS-specific behavior. This enables Diffusion clients to interact with data exchanged with a JMS server as Diffusion topics and data, without having to be aware of specifics of the JMS implementation.

The legacy JMS adapter v5.1 is deprecated and will be removed in a future release. We recommend you move to the new version.

For more information, see JMS on page 104.

Removal of message fragmentation

The message fragmentation capability has been removed from Diffusion.

Topic message fragmentation was intended to prevent head-of-line blocking by large messages. The API allowed messages for a given topic messages to be delivered out of order, which is incompatible with snapshot/delta processing.

If you applied topic message fragmentation to work around the maximum message size limitations, particularly for large topic load messages, we recommend instead that you increase the maximum message size to accommodate the largest possible application message.

Increasing the maximum message size to support large topic load messages will also require increasing the client input buffer and server output buffer sizes. The peak memory requirement is lower than needed when topic message fragmentation is enabled, but is approximately twice the maximum message size. In a future release, we will improve the buffer handling to allow the maximum message size to exceed the network buffer size.

Pooled input buffers

Changes to how input buffers are allocated improve the performance and memory usage of the Diffusion server.

Input buffers are no longer bound to clients, instead they are shared by all reading tasks. Where previously the number of connected clients defined the number of input buffers, now the maximum number of input buffers is bounded by the configured thread pool size.

The maximum amount of memory used for input buffers is less than the thread pool size multiplied by the input buffer size plus any small memory usage resulting from partial reads.

Non-exclusive updating

The Unified API now provides a simpler way for clients to update topics if the order of updates from different clients is unimportant. This is supported by the Java™, .NET, and JavaScript Unified API clients.

A client can register an update source against a topic. If the topic update source is active, it prevents updates to the topic from any other clients. However, a client can also update a topic without registering as an update source against it. If there is no update source registered against that topic, this non-exclusive update is successful. If multiple clients make non-exclusive updates to a topic, the most recent update is the one applied.

For more information, see Example: Make non-exclusive updates to a topic on page 274.

Windows and OSX support for the C Unified API

We provide libraries for the C Unified API that are compiled for Windows and OSX.

For more information, see C Unified API on page 467.

Proxy support (.NET Unified API)

Clients that use the .NET Unified API can now connect to the Diffusion server through a proxy. The Unified API enables you to connect through the proxy unauthenticated, with basic authentication, or through any other authentication process by implementing your own challenge handlers.

For more information, see Connecting through an HTTP proxy.

Flow control (.NET Unified API)

The .NET client assemblies can now control the flow of requests from a client to decrease the likelihood of the client's outbound queue or the client queue on the server overflowing and causing the client to be disconnected.

This process happens automatically when the client detects conditions that might cause a queue overflow and increases the reliability of the .NET client.

Related Links

Upgrading on page 25
If you are planning to move from an earlier version of Diffusion to version 5.5, review the following information about changes between versions.

What's new in Diffusion 5.1?

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

Key features

A complete list of the latest updates to Diffusion can be found in the Release Notes available at http://download.pushtechnology.com.

Paged topic support in the Java and .NET Unified API

You can use the TopicControl feature to create paged record topics and paged string topics and to define rule-based comparators to use for ordering the lines in the paged topic. For more information, see Creating paged topics on page 221.

You can use the TopicUpdateControl feature to update paged record topics and paged string topics. For more information, see Building updates for paged topics on page 262.

UpdateSource capabilities in the Unified API

The UpdateSource capabilities replace the TopicSource capabilities. UpdateSource includes the ability to build more complex updates, support for more topic types, and better handling of unexpected closes.

For more information, see the API documentation.

Remove topics after the control client closes using the Unified API

The TopicControl feature now enables you to specify whether to remove sections of the topic tree after a control client session closes.

For more information, see Removing topics with sessions on page 229.

.NET and C Unified API production support

The .NET and C Unified API are now supported for production use. Both APIs contain functionality to implement a control client.

For more information, see .NET Unified API on page 395 and C Unified API on page 467.

Proxy support

Clients that use the Java Unified API can now connect to the Diffusion server through a proxy. The Unified API enables you to connect through the proxy unauthenticated, with basic authentication, or through any other authentication process by implementing your own challenge handlers.

For more information, see Connecting through an HTTP proxy.

WebSocket support in the iOS® Classic API

Clients implemented using the iOS Classic API can now connect to the Diffusion server through the WebSocket protocol.

Streams replace listeners for receiving content through the Unified API

The `Listeners` in the Topics and Messaging features are now deprecated. Listeners have been replaced by streams. `Topics.TopicStream` receives topic events, such as topic updates for a topic or topics. `Messaging.MessageStream` receives messages sent through a topic or topics.

Streams provide advantages over listeners as a stream has a logical end. A stream can be closed or discarded, at which time the stream has the opportunity to do any required cleanup or take any required actions.

Flow control (Java Unified API)

The Java client library can now control the flow of requests from a client to decrease the likelihood of the client's outbound queue or the client queue on the server overflowing and causing the client to be disconnected.

This process happens automatically when the client detects conditions that might cause a queue overflow and increases the reliability of the Java client.

What's new in Diffusion 5.0?

The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

Key features

A complete list of the latest updates to Diffusion can be found in the Release Notes available at http://download.pushtechnology.com.

New high availability features

In version 5.0, Diffusion introduces the following new high availability features: session replication, topic replication, and failover of the active update source. These features use a datagrid to share data between multiple Diffusion servers.

Session replication shares client session information between servers. If a client loses connection to a server, it is reconnected through a load balancer to another server that has access to all of the client's session information.

Topic replication shares topic information – such as the topic definition and metadata – and topic data between servers. If a server becomes unavailable, the topic information and data is available on another server.

Only one server can act as the active update source for a topic or branch of the topic tree. If that server becomes unavailable, other servers can take over as the active update source for those topics.

For more information, see Replication on page 82.

Control client

Control clients are a way to package application logic that controls a Diffusion server. Unlike publishers, control clients run as a separate process outside of the server and use the Diffusion client library to communicate with the server.

Control clients use the Unified API to provide a secure remote control experience that can use all of the supported protocols to communicate with the Diffusion server. They can be implemented in any of the supported languages. For more information, see Control features on page 219.

Introducing the Unified API

Beginning in version 5.0, Diffusion is transitioning to a new public API. The Unified API will make available the capabilities of standard clients, control clients, and event publishers in one consistent, modular interface. For more information, see Unified API.

For version 5.0, the control features are now available. This enables you to replace remote control with the richer experience of control client.

The Classic API (the API used in version 4 and earlier) is still supported in 5.0 for clients and event publishers. The remote control API is no longer supported.

Improved performance

Diffusion can now serve up to 150% more messages per second to 60% more clients by using a new queuing mechanism.

In benchmark tests, using 50 topics and 125-byte messages, Diffusion served 15 million messages per second to 87,000 clients. Diffusion used 24 threads and three client processes to achieve this performance.

New authentication model

In Diffusion 5.5 we have split out the concept of authentication from that of authorization. You can write and configure both remote and local authentication handlers.

In previous versions, the authentication capability was provided by authorization handlers. Using authorization handlers for authentication is now deprecated. We recommend that you re-implement your authentication logic using the version 5 authentication APIs.

For more information, see User access control.

JavaScript API for paged topics

The JavaScript API now includes improved capability to work with paged topics. For more information, see Paged topic data in JavaScript on page 512.

Iframe streaming

Iframe streaming connections are now available over the HTTP protocol.

Liveness monitoring in Flex® and JavaScript

The Flex and JavaScript client libraries now include liveness monitors that listen for activity from the server and raise an event if the lack of activity indicates that the connection has been lost. This enables the client to reconnect in the event of a lost connection.

For more information, see Reconnecting with the ActionScript API on page 517 and Reconnecting with the JavaScript API on page 510

Upgrading

If you are planning to move from an earlier version of Diffusion to version 5.5, review the following information about changes between versions.

We recommend that you upgrade to the latest version of Diffusion as soon as you can.

When upgrading across multiple versions, ensure that you review the release notes and upgrade steps for all intermediate versions. For example, if you are upgrading from version 4.x to version 5.1, first follow the upgrade steps from version 4.x to 5.0, then follow the steps to upgrade from version 5.0 to 5.1.

Release notes are available at the following location: http://download.pushtechnology.com

For more information about Diffusion versions and support and upgrade policy, see the Support Center.

Related Links

What's new in Diffusion 5.5? on page 20
The latest version of Diffusion contains new functions, performance enhancements and bug fixes.

Interoperability

If you plan to use different versions of Diffusion servers and clients together, review the following information that summarizes support between versions.

Interoperation between clients and servers

The following table describes which client versions interoperate with which server versions, through the Classic API or Unified API:

	Server version			
Client version	**4.6**	**5.0**	**5.1**	**5.5**
4.6	YES	YES	YES	YES
5.0 Classic API	NO	YES	YES	YES
5.0 Unified API	NO	YES	NO	NO
5.1 Classic API	NO	YES	YES	YES
5.1 Unified API	NO	NO	YES	NO
5.5 Classic API	NO	YES	YES	YES
5.5 Unified API	NO	NO	NO	YES

Caution: For the 5.x release, we do not guarantee interoperability between clients and servers that use different 5.x versions of the Unified API. Be aware that when you upgrade between 5.x versions you might have to upgrade all servers and Unified API clients together.

Interoperation between servers

Publishers deployed to Diffusion servers can connect to and communicate with publishers deployed to Diffusion servers of different versions. The following table describes which server versions interoperate:

Server versions	4.6	5.0	5.1	5.5
4.6	YES	YES	YES	YES
5.0	YES	YES	YES	YES
5.1	YES	YES	YES	YES
5.5	YES	YES	YES	YES

Upgrading from version 4.x to version 5.0

Consider the following information when upgrading from Diffusion version 4.x to version 5.0.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

Remote control

The remote control APIs are no longer supported. Reimplement your remote control as a control client using the Unified API control features. For more information, see Control features on page 219 and Unified API.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see Interoperability on page 25.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation. Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided in the `docs` directory of your Diffusion installation
- The API documentation located at http://docs.pushtechnology.com/5.0

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.0 APIs, it fails. Rewrite your application code to not include these features.

Table 1: API features removed in version 5.0

API affected	Removed feature	Suggested alternative
Java API .NET API	Remote control	Reimplement your remote control applications as control clients using the Unified API. For more information, see Control features on page 219 and Unified API.
Java API	Methods in the `APIProperties` class: • `setInboundThreadPoolSize` • `getInboundThreadPoolSize`	Use the `ThreadsConfig` class instead. For more information, see Java Unified API documentation.
Android™ API	Methods in the `DiffusionClient` class: • getCredentials • setCredentials	Use the methods in `ServerDetails` or `ConnectionDetails` instead. For more information, see Android Classic API documentation.
Java API	`MessageComparator` interface and `compareTo` and `equals` methods on all Message classes.	Use conflation policies instead. For more information, see see Java Unified API documentation.
Java API	`TopicDetails` class	Use the `TopicDefinition` class instead. For more information, see Java Unified API documentation.
Java API	Methods in the `ThreadsConfig` class: • setWriterSelectors • getWriterSelectors	No longer used and no alternative required.
Java API	`Management`, `Proxy`, and `ServerProxy` interfaces	No longer used and no alternative required.
Java API	`Publisher.consoleLogLevelChange`	No longer used and no alternative required.
Java API	`ThreadServer.getOutboundThreadPool`	

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 2: API features deprecated in version 5.0

API affected	Deprecated feature	Suggested alternative
Java API	Using authorization handlers for authentication and the `AuthorisationHandler.canConnect` method.	Use authentication handlers instead. For more information, see User-written authentication handlers on page 145.
Java API	`APIProperties` class	Use methods in the `Utils` or `RootConfig` classes instead. For more information, see Java Unified API documentation.
Java API .NET API	Event publishers	Reimplement your event publishers as control clients using the Unified API. For more information, see Control features on page 219 and Unified API.
Java	`Client.getNumberOfMessagesSent` and `Client.getNumberOfMessagesReceived`	Use `Client.getStatistics` instead. For more information, see Java Unified API documentation.
Java	Methods that navigate up from a configuration item to its parent configuration item.	Instead navigate down from the root configuration item. For more information, see Java Unified API documentation.
Java	`MNode.getMessage`	No longer used and no alternative required.

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- The Publisher API methods that add topics no longer block until automatic pre-emptive subscriptions have been processed. Matching pre-emptive subscriptions are be completed in the background.
- The Java API now enables you to set auto-subscribe using a `TopicDefinition`
- The format of the generated client IDs has changed
- `getStatistics` no longer returns null if statistics recording is disabled. Instead it returns a value of -1.
- Clients that subscribe to topics that they are already subscribed to, no longer receive an initial topic load.

Upgrading your server installation

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

   For more information, see Installing on page 560.
2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.

3. You can copy your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:

   - In the `WebServer.xml` configuration file for your production installation, remove or comment out the configuration for the HTTP deploy service.

     Access to this service is not restricted. If you enable the deploy service, you must restrict access to the deploy URL by other methods to prevent unauthorized or malicious access. For example, by setting up restrictions in your firewall.
   - Remove the `writer-selector` configuration from the `Server.xml` configuration file. Writer selectors are no longer used.

     Warning: Do not confuse writer selectors with write selectors.
   - If you now use authentication handlers for authentication, configure these handlers in the `Server.xml` configuration file.
   - If you now use the replication high availability features, configure these in the `Replication.xml` configuration file.
   - If you use the WhoIs service, but do not explicitly configure it, you must now configure the service in the `Server.xml` configuration file.

     In previous releases, if no configuration was specified for the WhoIs service, the service started with the default configuration. In this release, the service does not start unless configuration is present in the `Server.xml` configuration file.

   The validation of the configuration files has been relaxed. The order of the element within the files is less strict.
4. If you start the Diffusion server from your own scripts or Java programs, you must update them to take into account the following changes:

   - The Java license agent has been removed. Remove the following argument from the Java command you use to start the server:

     ```
     -javaagent:../lib/licenceagent.jar=../etc/licence.lic,../
     etc/publicKeys.store
     ```
   - New system properties are required by the Diffusion server.

     Include the following properties in the Java command that starts the server:

     ```
     -Ddiffusion.license.file=diffusion_installation/etc/
     licence.lic
     -Ddiffusion.keystore.file=diffusion_installation/etc/
     publicKeys.store
     -Ddiffusion.home=diffusion_installation/lib
     ```

     You can also supply these properties as VM arguments.

     For more information, see Running from within a Java application on page 650

   If you use the start scripts provided with the Diffusion installation, you do not need to make any changes.

Upgrading from version 5.0 to version 5.1

Consider the following information when upgrading from Diffusion version 5.0 to version 5.1.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

Event publishers

The event publisher APIs are deprecated. Reimplement your event publisher as a control client using the Unified API control features. For more information, see Control features on page 219 and Unified API.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see Interoperability on page 25.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation. Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided in the `docs` directory of your Diffusion installation
- The API documentation located at http://docs.pushtechnology.com/5.1

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.1 APIs, it fails. Rewrite your application code to not include these features.

Table 3: API features removed in version 5.1

API affected	Removed feature	Suggested alternative
Publisher API Event Publisher API	The capability to set the maximum queue size to -1, which specified an unbounded queue size, using `Client.setMaximumQueueSize()`.	Set the maximum queue size value to a positive integer.

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 4: API features deprecated in version 5.1

API affected	Deprecated feature	Suggested alternative

Unified API	`TopicUpdateControl.TopicSource`
	`TopicUpdateControl.TopicSourceDefault`
	`TopicUpdateControl.TopicSourceUpdater`
		`TopicUpdateControl.UpdateSource`
		`TopicUpdateControl.UpdateSource.Default`
		`TopicUpdateControl.Updater`

Java Unified API	`Messaging.Listener` and associated methods	`Messaging.MessageStream`
Java Unified API	`Topics.Listener` and associated methods	`Topics.TopicStream`
Java Unified API	`Updater.update()` methods that take both `Content` and `UpdateOptions` as parameters	`Updater.update()` methods that take `Update` as a parameter
Java Unified API	`comparator()` and `duplicatesPolicy()` methods in the `PagedTopicDetails.Builder` class	`order(String)`, `order(Duplicates, String)`, or `unordered()`
Java Unified API	`getComparator()` and `getDuplicatesPolicy()` methods in the `PagedTopicDetails.Attributes` class	`getOrderingPolicy()`
.NET Unified API	All `Set`*`Property`*`()` methods in the `ISessionFactory` class, where *Property* is the name of the value you want to change	*`Property`*`()`
Unified API	The `autoSubscribe` method in the `TopicDetails.Builder` interface. In future, auto-subscribe is always true.	None
JavaScript Classic API	The functions `setCrypted()` and `getCrytped()`	`setEncrypted()` and `isEncrypted()`
Java Server API	The functions `getStartTimeMillis()`, `getUptime()`, and `getUptimeMillis()` on the `c.p.d.api.topic.Subscription` class	You can use a publisher to get equivalent functionality.

API affected	Deprecated feature	Suggested alternative
Java Unified API	All static fields in the `c.p.d.client.types.Constants` class	These fields are now available in the `c.p.d.client.content.Record` class.
Java Unified API, .NET Unified API	`RecordContentReader.hasMore()`	`RecordContentReader.hasMoreRecords()`, `RecordContentReader.hasMoreFields()`

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- The UpdateSource API that replaces the TopicSource API in the Unified API behaves in a very similar way, but has some differences.
  - UpdateSource includes an `onRegister` callback, which provides a `RegisteredHandler` that the client can use to deregister as an update source. Previously, TopicSource only provided the ability to deregister as an update source to the client that was the active update source.
  - UpdateSource includes an `onError` callback, which indicates to a client when an update source is prematurely closed.
  - Updaters do not accept an `UpdateOptions` object as a parameter. Instead an `Update` object is used to contain the update content and any additional information about the update.

  For more information about how update sources work, see Updating topics on page 260.
- In the Unified API, the logging level at which the `Topics.Listener.Default` logs updates has changed from “warn” to “debug”.
- In the Unified API, topic and messaging listeners are called in the order that they were registered. In previous releases, these listeners might have been called in any order.
- In the Unified API, you must set a fallback topic or messaging listener explicitly. In previous releases, a fallback topic or messaging listener was set by default.
- In the Unified API, a notification occurs for any type of selector that does not match with any topics. In previous releases, a missing topic notification occurred only if a topic path selector was used for subscribe or fetch and there was no such topic.
- In the Unified API, you can add multiple session listeners and remove session listeners. In previous releases, you could add only one session listener.

Upgrading your server installation

Note: At release 5.1, the Diffusion server is tested and supported on Java HotSpot™ Development Kit 8 (latest update). If you are using an earlier update of Java 7, we recommend that you update to update 67. This update includes a number of security improvements over the previously supported version of Java 7.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

   For more information, see Installing on page 560.
2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.

3. You can copy your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:
    - In the `Server.xml` and `Connectors.xml` configuration files, if you have set maximum client queue depths to be unbounded (0), change these values. Unbounded outbound client queues are no longer allowed.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- In previous releases, messages that required acknowledgment were prioritized over other messages. This might have caused ordering problems. From 5.1, messages that require acknowledgment are queued for sending in the order of receipt. You might have to increase your acknowledgment timeout value to allow for the additional queuing time.
- The non-configurable 5s timeout between HTTP polls has been removed. Use `<system-ping-frequency>` for HTTP connectors.

Upgrading from version 5.1 to version 5.5

Consider the following information when upgrading from Diffusion version 5.1 to version 5.5.

Upgrading your applications

Server-side components

Recompile all Java application components that are deployed to the Diffusion server, such as publishers and authorization handlers, against the new version `diffusion.jar` file. This file is located in the `lib` directory of your new Diffusion server installation.

Some features that your Java application components might use have been removed or deprecated. Pay attention to new deprecation warnings and compilation failures that occur during recompilation and review the API changes information in the following section to see if these changes affect your applications.

Event publishers

The event publisher APIs have been removed. Reimplement your event publisher as a client using the Unified API control features. For more information, see Control features on page 219.

JMS adapter

The legacy JMS adapter has been deprecated and replaced with a new version. These JMS adapters are not compatible.

To move to the latest JMS adapter, use the `JMSAdapter.xml` configuration file to configure the behavior. Refer to the provided XML and XSD file because the configuration has changed since the previous version. For more information, see JMSAdapter.xml.

> **Note:** Not recommended: To continue using the legacy JMS adapter, version 5.1, rename the `JMSAdapter.xml` configuration file used with the previous versions of Diffusion to `JMSAdapter51.xml`.

Clients

You can choose not to recompile your client applications and continue to use client libraries from a previous release. If you choose to use client libraries from a previous release, ensure that the libraries are compatible with the new server. For more information, see Interoperability on page 25.

You can choose to upgrade your client applications to use the new client libraries. To do this, recompile the client applications against the client libraries located in the `clients` directory of your new Diffusion server installation and repackage your client application with the new library.

Alternatively, the Java library is available in the Push Public Maven Repository and the JavaScript library from NPM.

Some features that your client applications might use have been removed or deprecated. Review the API changes information in the following section to see if these changes affect your applications.

> **Note: .NET clients**: When you recompile your .NET clients with the new version of the libraries, be aware that the latest .NET client libraries require version 4.5 or later of the .NET Framework. In addition, only a single Diffusion DLL is now required to compile a Diffusion .NET client. For more information, see .NET Unified API on page 395.

API changes

Further information about removed or deprecated features is available in following locations:

- The release notes provided in the `docs` directory of your Diffusion installation or online at http://docs.pushtechnology.com/docs/5.5.1/release/ReleaseNotice.html
- The API documentation located at http://docs.pushtechnology.com/5.5

The following table lists API classes and methods that have been removed. If you attempt to recompile application code that uses these classes or methods against the version 5.5 APIs, it fails. Rewrite your application code to not include these features.

Table 5: API features removed in version 5.5

API affected	Removed feature	Suggested alternative
Event Publisher API	All	Use the Unified API.
Java API	`getConflation` method on `RootConfig`	`getConflation` method on `ServerConfig`
.NET Classic API	`Credentials`	`V4Credentials` This change was made to disambiguate between the credentials object in the Classic API and that in the Unified API.

The following table lists API classes and methods that have been deprecated. If your application code uses these classes or methods, consider rewriting your application code to not include these features.

Table 6: API features deprecated in version 5.5

API affected	Deprecated feature	Suggested alternative
Java Unified API, .NET Unified API, C Unified API	`Session.start()` and the associated 'initialising' state. Now a no-op.	The asynchronous 'open' method on `SessionFactory` which has callback to notify session opened (or error).
Java Classic API	`Client.setCredentials()`, `Client.getCredentials()`	None
Publisher API	`AuthorisationHandler.credentialsSupplied()`	None
Java Unified API	`TopicSubscriptionHandler`	Use routing topics instead
Publisher API	`XMLPropertiesListener`, `XMLProperties`	None
Publisher API	`MultiplexerConfig.getLoadBalancer()` and `MultiplexerConfig.setLoadBalancer()`	None These methods enable you to specify the load balancing policy for multiplexers. Previously, round robin and least clients policies were available. In future, only the default, round robin, policy will be provided.
Java API	`com.pushtechnology.com.api.config.ManagementUserConfig` and related methods in `com.pushtechnology.com.api.config.ManagementConfig`	Configure the system authentication store or a custom authentication handler to manage JMX users.
.NET API	`Session.getFetchFeature` and the `Fetch` feature	`Session.getTopicFeature`. The fetch capabilities are included in the Topics feature.
.NET API, Java API	`Headers` interface	`ReceiveContext.getHeaderList()` and `SendOptions.headers(List<String>)`. Headers are now represented in the API as a list of Strings. .
Java API	`RootConfig.getMessageLengthSize`, `RootConfig.setMessageLengthSize`	These methods are both deprecated and no-ops. The message length size is now hardcoded to 4 bytes.
Publisher API	`TopicProperty.AUTO_SUBSCRIBE` property. This is ignored. Auto-subscribe is always true.	None

API affected	Deprecated feature	Suggested alternative
Configuration API	`WriteSelectorConfig` This is ignored. Write selectors have been unified with other types of selector.	`SelectorThreadPoolConfig` All selectors are now drawn from the selector thread pool.

The following list includes behavior that has changed in the API. If your application code relies on the previous behavior, rewrite your application code to take into account the new behavior.

- You can now create a `SessionId` object from the session ID String in the Java Unified API and .NET Unified API.
- The result of calling the `AuthorisationHandler.canWrite()` is no longer cached. If an authorisation handler is registered, it is called every time a client sends a message to the server.

  Note: We recommend you use role-base security instead of authorisation handlers.
- Auto-subscription is enabled for every topic and cannot be disabled. If a client attempts to subscribe to a topic that does not exist, the subscription request is saved and when the topic is created, the client is automatically subscribed to the topic.
- The `Publisher.subscription()` method no longer sends a load message for a topic that has state. When a topic with state - that is, a topic whose data type is not `TopicDataType.NONE` - is first subscribed to, the topic load message is sent before `Publisher.subscription()` is called. The default implementation of the `Publisher.subscription()` method does nothing.
- Topic loaders are only called for topics that have no state - that is, topics whose data type is `TopicDataType.NONE`.
- `SlaveTopicData` no longer extends `PublishingTopicData`.
- Updates to slave topics no longer update the master topic.
- Changes to how the number of subscribers to a topic are counted:
  - Slave topics only count subscriptions made directly to the slave topic.

    In previous releases, subscriptions to the master topic and other slave topics of the same master were counted as subscriptions to a slave.
  - Master topics count all subscriptions made directly to the master topic and all subscriptions made indirectly through slave topics.
  - Topics subscribed to through a routing topic count both direct subscriptions to the topic and indirect subscriptions through the routing topic.

  These changes affect the return values from methods that query whether a topic has subscribers or the number of topic subscribers.
- Classic API clients are no longer required to be subscribed to topic paths that they send messages on. However, a topic must exist at the topic path for a Classic API client to receive a message through the topic path.

  Classic API clients are still required to be subscribed to topic paths to receive messages on those topic paths.

Upgrading your server installation

Note:

At release 5.5, the Diffusion server is tested and supported on Java HotSpot Development Kit 8 (latest update).

The Diffusion server also runs on Java HotSpot Development Kit 7. However, Oracle® withdrew support for Java 7 in April 2015. We recommend that you move to the latest update of Java 8 as soon as possible.

To upgrade your Diffusion server installation, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

   For more information, see Installing on page 560.
2. You can copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. You can copy most of your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation. When you do, consider making the following changes:

   - The structure of `JMSAdapter.xml` file has changed. Do not copy your existing `JMSAdapter.xml` file to the `etc` directory of your new installation. Instead copy it to `etc/JMSAdapter51.xml`. This change is because the legacy JMS adapter is deprecated and is replaced with a new version. For more information, see Configuring the JMS Adapter
   - The `SubscriptionValidationPolicy.xml` configuration file is now deprecated. Use the roles and permissions provided in the new security model to define which clients can subscribe to which topics.
   - Configuring remote JMX users in the `Management.xml` configuration file is now deprecated. Instead use the system authentication store or a custom authentication handler to configure remote JMX users.
   - When configuring log levels in the `Logs.xml` configuration file, use the SLF4J log levels: ERROR, WARN, INFO, DEBUG, or TRACE. The `java.util.Logging` log level values (SEVERE, WARING, INFO, FINE, and FINEST) are still supported, but are deprecated.
   - You can now configure the format of the date in log file names in the `Logs.xml` configuration file by using the optional `<date-format>` element.
   - Configuring the multiplexer load balancing policy in the `Server.xml` configuration file is now deprecated. In future, only one load balancing policy will be provided for multiplexers: the default, round robin, policy.
   - In the `Server.xml` configuration file, when defining the file name of the GeoIP database file you must use an absolute path or a path relative to the Diffusion installation directory. Backwards compatibility with version 4.6 has been removed. You can no longer specify a path relative to the configuration directory.
   - The connector for port 8081 has been removed from the `Connectors.xml` configuration file. If your clients connect on this port, you must either configure this port in `Connectors.xml` or change the port that your clients connect on.
   - You are now required to configure a selector thread pool definition. Update your `Server.xml` configuration file to ensure that it includes this definition.
   - The write selector configuration in the `Server.xml` configuration file is deprecated and any configuration associated with write selectors is ignored. All selectors are now drawn from the same pool. Use the selector thread pool definition elements to define the number and behavior of selectors.

Behavior changes at the Diffusion server

The following list includes behavior that has changed at the server. If your solution relies on the previous behavior, adjust your solution to take into account the new behavior.

- The message fragmentation capability has been removed from Diffusion.

Topic message fragmentation was intended to prevent head-of-line blocking by large messages. The API allowed messages for a given topic messages to be delivered out of order, which is incompatible with snapshot/delta processing.

If you applied topic message fragmentation to work around the maximum message size limitations, particularly for large topic load messages, we recommend instead that you increase the maximum message size to accommodate the largest possible application message.

Increasing the maximum message size to support large topic load messages will also require increasing the client input buffer and server output buffer sizes. The peak memory requirement is lower than needed when topic message fragmentation is enabled, but is approximately twice the maximum message size. In a future release, we will improve the buffer handling to allow the maximum message size to exceed the network buffer size.

The default value of the input buffer size has been increased to 1M.

- The default number of multiplexers has changed from 2 to the number of available processors.
- Changes to how input buffers are allocated improve the performance and memory usage of the Diffusion server.

  Input buffers are no longer bound to clients, instead they are shared by all reading tasks. Where previously the number of connected clients defined the number of input buffers, now the maximum number of input buffers is bounded by the configured thread pool size.

  The maximum amount of memory used for input buffers is less than the thread pool size multiplied by the input buffer size plus any small memory usage resulting from partial reads.
- It is possible to define a set of selector thread pools and have a connector refer to a member of this set name. If no pool is defined, each connector is assigned a default pool with size = 1.
- Reconnection is enabled by default, with a keep-alive time of 60 seconds. If you do not want client sessions to be able to reconnect, disable reconnect in the `Connectors.xml` configuration file.

  When reconnection is enabled, the Diffusion server continues to queue messages for a client session for the whole keep-alive time. This can affect performance.
- The `etc` directory of your Diffusion installation is no longer on the classpath. If you have included any files in the `etc` directory that you require to be on the classpath, ensure that these files are included in the classpath by other means:
  - Place resource files such as `hazelcast.xml` that must be on the JVM system classpath in the `data` directory of the Diffusion installation.
  - Place resource files that are only loaded by deployed application code, such as publishers or authentication handlers, in the `ext` directory of the Diffusion installation.
- Wrapper scripts for `jstatd` and `jstack` are no longer provided in the `tools` directory of your Diffusion server installation.
- Messages sent on topic paths are no longer counted in the topic statistics. Only updates to the topic are counted.
- The message length size is no longer configurable. In all cases it is hardcoded to 4 bytes. If you have earlier versions of the Diffusion server as part of your solution, ensure that their message length size is configured to be 4 bytes or the servers will not interoperate.
- The names of the keystores provided in the Diffusion installation have changed.
  - publicKeys.store is now licence.keystore
  - keystore is now sample.keystore

  The start scripts provided by the Diffusion installation have been updated accordingly. If you use these scripts, you do not need to make any changes.

If you start the Diffusion server from your own scripts, you must update them to take into account this change. Update the following property in the Java command that starts the server to point to the new keystore name:

```
-Ddiffusion.keystore.file=diffusion_installation/etc/
licence.keystore
```

If you start the server from a Java program and supply the keystore name as a VM arguments, update the VM to point to the new keystore name. For more information, see Running from within a Java application on page 650

- The supported version of Google protocol buffers is now 2.6.1. In previous versions it was 2.4.1.
- Due to the addition of the new JavaScript Unified API, the JavaScript API you might have been using with previous versions of Diffusion is now called the Classic API.

  The library for the JavaScript Classic API is now called `diffusion-js-classic-version.js` and the API documentation for it is now located in the `js-classic` folder of the documentation.

Upgrading to a new patch release

When upgrading to a new patch release there are typically no changes to the configuration values or the APIs. All that is required is to copy your existing files from the old installation to the new installation.

To upgrade to a new patch release, complete the following steps:

1. Use the graphical or headless installer to install the new version of Diffusion.

   For more information, see Installing on page 560.
2. Copy your existing license file from your previous installation to the `etc` directory of your new installation.
3. Copy your existing configuration files from the `etc` directory of your previous installation to the `etc` directory of your new installation.
4. Copy any publishers located in the `ext` directory of the previous installation into the `ext` directory of the new installation.

Part II

Design Guide

This guide describes the factors to consider when designing your Diffusion solution.

In this section:

- Overview
- Designing your data model
- Designing your solution
- Security

Overview

Diffusion is a component-based and modular data distribution platform.

Some Diffusion components are optional. Only the Diffusion server, publishers, and clients are required to create a working platform. One of the benefits of component-based architecture is the ability to build your architecture as your business requirements grow.

This section provides an overview of the components and functionality of the Diffusion platform.

Architecture

Diffusion is a data distribution platform that uses a low latency messaging server to merge and distribute data from multiple sources to client applications.

The following diagram shows the components of a Diffusion architecture and how those components interact.

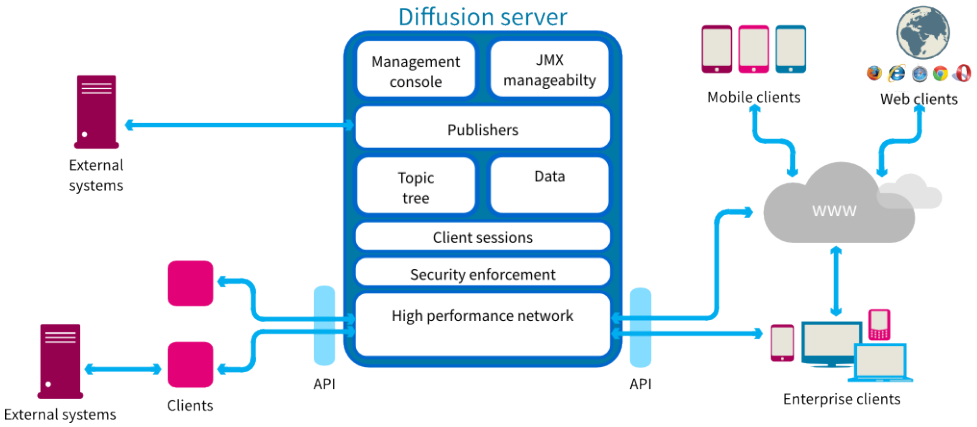


Figure 1: Basic architectural components

The following components can make up a basic Diffusion architecture:

Diffusion server

You must have at least one Diffusion server in your architecture.

The Diffusion server hosts publishers and the topic tree. It manages connections from clients and pushes data to the clients through message queues.

Publisher

Publishers create and maintain topics. They coordinate the distribution of content on topics. Publishers are written in Java and deployed within a Diffusion server.

Client

Client applications connect to the server.

The diagram shows different categories of client: web, mobile, and enterprise. The category of client depends on the language of the API and libraries used to implement it. Clients can be implemented in one of a number of languages and use variety of protocols to communicate with the server.

Clients can perform different types of actions depending on their permissions and the capabilities of the API they use.

Clients used by your customers typically subscribe to topics and receive from the server the updates that are published to these topics.

Clients used by your organization typically create, manage, and update topics. These clients also take responsibility for control functions, for example authenticating and managing other client sessions.

The Diffusion APIs

The Diffusion APIs and associated client libraries come packaged as part of the server. They are provided for a number of different platforms. Clients must implement the API and include the libraries appropriate to their platform to interact with the server.

Example of a scalable, resilient architecture

Your Diffusion architecture can use the following features to be scalable and resilient:

- Load can be spread across different publishers on different servers.
- Publishers can act as clients by subscribing to topics on other servers.
- Servers can share information with each other by replicating it into a datagrid.
- Servers can load balance requests across multiple clients that have registered to handle the request.

The following diagram shows how these features can be used in an architecture.

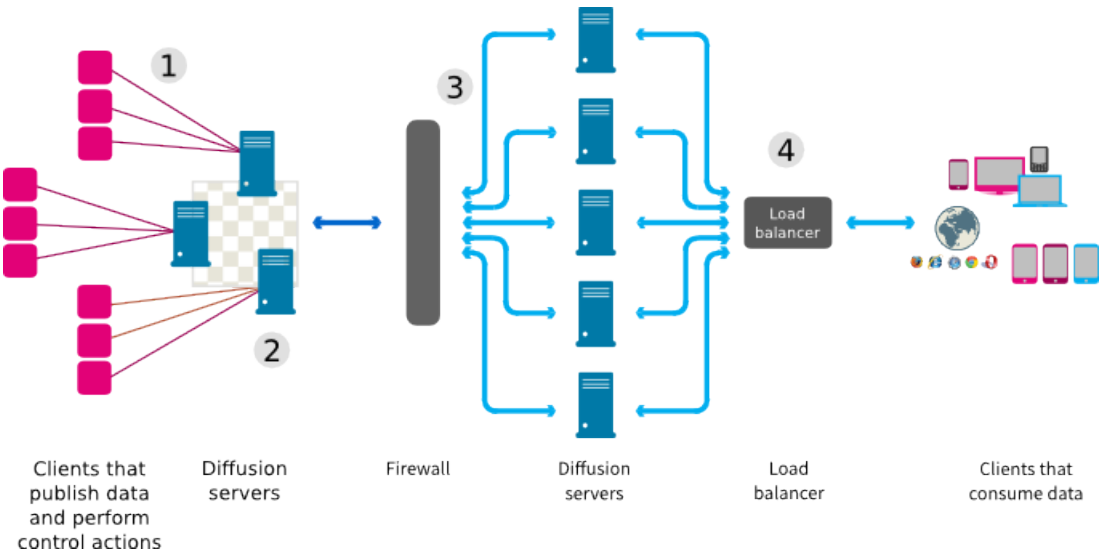


Figure 2: A scalable, resilient architecture

1. Three clients register handlers with each of the Diffusion servers behind the firewall. These clients can be located on the same system as the server or on remote systems. The server load balances requests between clients that have registered to handle requests of that type. If one of the clients becomes unavailable, the requests can be directed to another client. You can connect more client sessions to deal with higher volumes of requests.

2. The Diffusion servers inside the firewall replicate information into a datagrid. If a server that was handling a client session or topic becomes unavailable, the responsibility for that client session or topic can be passed to another server that has access to all the information for that session or topic through the datagrid.
3. Publishers on the Diffusion servers outside of the firewall, in the DMZ, can subscribe as clients to the topics on the Diffusion servers inside the firewall.
4. You can use a load balancer to spread requests from subscribing clients across many Diffusion servers. If a server becomes unavailable, clients can be directed to another server.

Related Links

Diffusion server on page 44
The Diffusion server is a highly efficient, low latency messaging server that can be deployed as a standalone server or as part of a cluster to provide a fully scalable enterprise data distribution solution.

Publishers on page 45
Publishers are components hosted within a Diffusion server that manage the data for one or more topics and publish messages to any clients that subscribe to the topics that the publisher manages.

Clients on page 47
A client is any application that communicates with the Diffusion server using a Diffusion client protocol.

Diffusion APIs on page 48
Diffusion provides application programming interfaces (APIs) that you can use to create applications that interact with the Diffusion server.

Data distribution

Diffusion provides a high performance infrastructure for distributing data. There are two mechanisms by which Diffusion distributes data: by publishing messages to topics or by sending messages between applications.

What is a message?

Data sent through Diffusion is formatted in messages. A message is a contiguous sequence of bytes comprising some header information followed by a data payload.

There is no limit on the format of data held within a message. Messages can be encoded to either compress or encrypt the data content of the message.

Messages are sent between server, clients and other components over socket-based connections using the Diffusion message protocol.

Publishing to topics

The primary mechanism of distributing data through Diffusion is by using topics to provide pub-sub messaging.

Publishers and clients can create topics on the Diffusion server and publish messages to topics.

Clients can subscribe to topics and receive messages that are published to that topic.

You can use this mechanism to send from many sources to many recipients.

Sending messages

Data can be distributed between endpoints by sending messages through Diffusion.

Publishers and clients can send messages to clients or groups of clients.

Clients can send messages to a topic path. The Diffusion server routes those messages to the publisher that owns the topic path or the client that is registered to handle messages sent on that topic path.

Related Links

Messages

Diffusion server

The Diffusion server is a highly efficient, low latency messaging server that can be deployed as a standalone server or as part of a cluster to provide a fully scalable enterprise data distribution solution.

The Diffusion server provides the following main functions:

- A scalable and mature solution to push (stream) and receive data and events, in real-time, both to and from client applications.
- High performance transport of data between two or more application within your own network or extranet.

The Diffusion server has been certified to run on the Oracle Java HotSpot Development Kit 8 (latest update).

The Diffusion server also runs on Java HotSpot Development Kit 7. However, Oracle withdrew support for Java 7 in April 2015. We recommend that you use the latest update of Java 8.

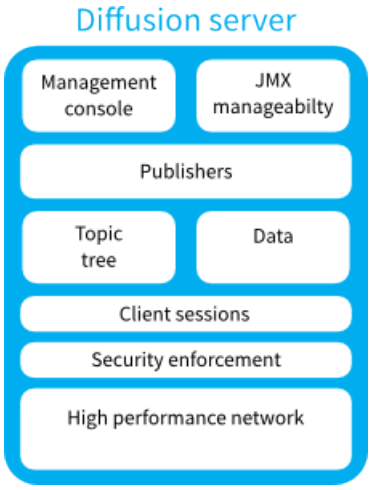


Figure 3: Components in a Diffusion server

High performance network layer

The high performance network layer uses the Java NIO framework to handle a high number of concurrent connections without the need for separate threads. Connectors handle connections from many different types of client and on many protocols. You can configure many connectors to listen on different ports. Multiple clients can connect to a single port.

Security enforcement

The Diffusion server authenticates all connections from clients. It also manages authorization and permissioning for actions that those clients can take when they are connected to the server.

Client sessions

The server manages the sessions for all of the clients that connect to it. It stores information about the client and the client's subscriptions. If a client disconnects, it can reconnect to the same session within a certain time period.

Topic tree

Topics are arranged in a tree structure. Inside the topic tree are top-level topics with subordinate topics. These subordinate topics can themselves have subordinate topics. The topic tree is a model of the data that is published to clients.

Data management

The Diffusion server performs operations on the data to more efficiently deliver it to clients. It performs structural conflation, merging and replacing data to ensure that the latest data is received by the client.

Publishers

Publishers are user-written applications that are hosted and run within the Diffusion server. Publishers are written in Java. They create and maintain topics.

Management console

Diffusion provides console as an optional publisher that is deployed by default. You can use it to monitor the operations of your Diffusion server through a web browser and to stop and start publishers within your server.

JMX manageability

Diffusion registers MBeans for many of its principal features with the JMX service. You can use a JMX management console, such as JConsole, to manage the Diffusion server.

Related Links

Architecture on page 41
Diffusion is a data distribution platform that uses a low latency messaging server to merge and distribute data from multiple sources to client applications.

Server on page 648
The Diffusion server is the core component of the Diffusion Product.

Publishers

Publishers are components hosted within a Diffusion server that manage the data for one or more topics and publish messages to any clients that subscribe to the topics that the publisher manages.

Publishers are written using the Java API and must extend the issued `Publisher` class and implement various methods to provide the publisher functionality

A publisher must maintain its own data model. The publisher can initialize its data as it starts and update it as a result of external events.

When a client first subscribes to a topic the publisher can provide the client with a snapshot of the current state of the data relating to that topic. This is called a topic load.

A client can also request the current state of a topic, even if not subscribed to it, using the fetch command. There is even the possibility of a publisher responding to a fetch request of a topic that does not exist. This provides a potential request/response mechanism without the overhead of real topics.

A publisher must maintain any changes to its topic data state and publish those changes to the topic as delta messages. This results in the message being sent to every client that is subscribed to the topic.

Publishers can send messages to individual clients or to groups of clients and can receive messages from clients.

For normal publishing the publisher does not need to know or keep track of the clients subscribed to its topics.

Publishers own the topics they create. Ownership of a topic is used to determine which publisher receives a message from a client, deals with subscription, and or creates dynamic topics.

Publisher clients

Publishers hosted in a Diffusion server can act as client applications to other Diffusion servers. A publisher can do this by subscribing to topics on the other server. This type of communication enables you to create a distributed Diffusion architecture.

Related Links

Architecture on page 41
Diffusion is a data distribution platform that uses a low latency messaging server to merge and distribute data from multiple sources to client applications.

Publishers

Topics

Clients and publishers or control clients are loosely coupled through logical links called topics.

A publisher or control client publishes messages to a topic and a client subscribes to a topic and receives its messages.

A topic path can also be used by a client to send messages to the publisher or control client that receives messages on that topic path. The client is not aware of the publisher or control client, only of the topic path.

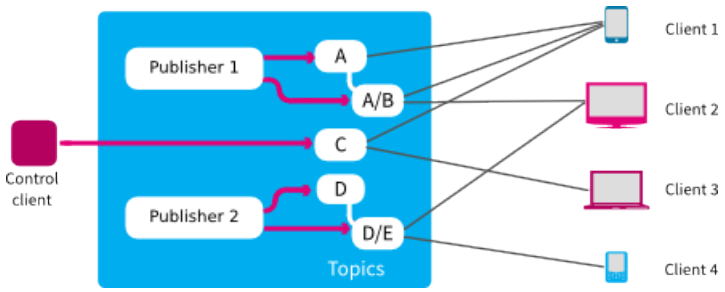


Figure 4: Topics

Topics are created in a Diffusion server by publishers or control clients. Each topic must have a unique name within the server. In the diagram the topic names are A, B, C, D, and E.

Topics can be arranged in a tree structure. In the diagram topic B is beneath topic A in the topic tree and topic E is beneath topic D in the topic tree.

The location of the topic in the topic tree is described by the topic path. The topic path includes all the topics above it in the topic tree in order separated by the slash character (/). In the diagram the path to topic B is A/B and the path to topic E is D/E.

The topic tree can include any number of topics. There are no limits to the number of topics a control client or publisher can update. There are no limits to the number of topics a client can subscribe to.

Related Links

Topics on page 51
Consider the types of topic you want to use and how. You can also consider the attributes that topics have. The attributes a topic can have can change depending on the topic type.

Clients

A client is any application that communicates with the Diffusion server using a Diffusion client protocol.

Most clients connect to the Diffusion server only to subscribe to topics and receive message data on those topics. Some clients, which have more permissions, perform control actions such as creating and updating topics or handling events.

Some of the types of client that are supported by Diffusion are listed in the following sections.

Web clients

Clients can be browser applications using one of the following APIs provided by Diffusion:

- JavaScript
- ActionScript
- Silverlight®

Enterprise clients

A client can be any application connecting to the Diffusion server over a DPT socket connection which can be over the internet or an intranet/extranet. Enterprise clients can use one of the following APIs provided by Diffusion:

- Java
- .NET
- C

Applications in other languages can communicate using the raw DPT.

Mobile clients

Clients can be mobile applications using one of the following APIs provided by Diffusion:

- iOS
- Android

Related Links

Architecture on page 41

Diffusion is a data distribution platform that uses a low latency messaging server to merge and distribute data from multiple sources to client applications.

Clients

Diffusion APIs on page 48
Diffusion provides application programming interfaces (APIs) that you can use to create applications that interact with the Diffusion server.

Diffusion APIs

Diffusion provides application programming interfaces (APIs) that you can use to create applications that interact with the Diffusion server.

The Unified API

You can use the Unified API to write clients and control clients. The Unified API is a modular, feature-based API and is available for JavaScript, Java, .NET, and C.

The Classic API

The Classic API comprises a number of APIs.

Client APIs

You can use the client APIs to create clients that subscribe to topics and receive messages from topics. The client API is available for the following platforms: Java, .NET, JavaScript, ActionScript, Silverlight, iOS, Android, and C.

Publisher API

You can use the publisher API to create a publisher that runs inside the Diffusion server. A publisher must extend the `Publisher` class. The publisher API is available in Java.

Related Links

Architecture on page 41
Diffusion is a data distribution platform that uses a low latency messaging server to merge and distribute data from multiple sources to client applications.

Table of Classic APIs on page 497
Diffusion provides Classic APIs in a variety of languages. Each of these APIs is supported over a set of protocols and in a specific level of the language.

Designing your data model

Distribute your data in a data model that fits the needs of your organization and customers.

There are a number of things to consider when designing your data model:

- The structure of your topic tree
- The types of topic to use
- The format of your data
- How you publish data to topics
- Your conflation strategy
- Whether you also use messaging to send data.

These considerations are not separate. The decisions you make about one aspect of your data model affect other aspects.

The data model is defined on the Diffusion server by your publishers or clients. The topic structure, topic types, and data format are not persisted on the Diffusion server through a restart or upgrade.

Design your solution to create your data model on the Diffusion server afresh after the Diffusion server is restarted or upgraded.

Topic tree

Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

What is the topic tree?

The topic tree is the organizational structure of the Diffusion topics. A topic of any type can be bound to any node of the topic tree. You do not have to explicitly bind a topic to every node of the topic tree.

Locations in the topic tree are referred to by their topic path, which is the level names in the tree that lead to the topic, separated by the slash character (/).

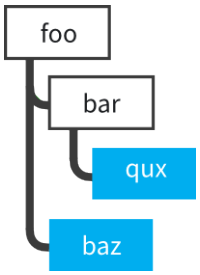


Figure 5: Example topic tree

In the preceding image, topics are bound to baz and qux. The topic path for baz is /foo/baz. The topic path for qux is /foo/bar/qux

You can create a topic and bind it to /foo/bar/qux without having to create and bind topics at /foo or /foo/bar beforehand.

There can be multiple topics that have the same name, but topic paths are unique.

When interacting with topics in the topic tree, your clients can reference a topic by its topic path or by a *topic selector* with a pattern expression that matches the topic path. Clients can use topic selectors to reference sets of topics in the tree that all match the topic selector's pattern expression.

Considerations when designing your topic tree

- Does the source information have a logical organization?

  If the data structure of the source information is complex, it can be mapped to a hierarchical structure.
- How many topics?

  If the number of topics is small, a flat topic tree might be appropriate.
- How do clients subscribe to the data?

If there are topics that clients generally subscribe to together, these topics can be organized in the same branch of the topic tree. This enables a client use a topic selector to subscribe to the branch of the topic tree and receive updates for all topics in that branch.

- The size of your topic tree can be constrained by your hardware.

  An extremely large topic tree can cause long GC pauses. Ensure that you do sufficient testing with your topic tree before going to production.

  If the size of your topic tree structure is caused by a lot of duplication, use routing topics to reduce it.

Related Links

Topic selectors in the Unified API on page 66
A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topic selectors in the Classic API on page 72
A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Topic selectors in the Unified API on page 66
A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topic selectors in the Classic API on page 72
A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Topic naming

Consider the following restrictions when deciding on your topic names.

Restricted characters

A topic name can be made up of any number of Unicode characters but must not contain any of the restricted characters mentioned below. The topic path is made up of the names of all topics in its path separated by the slash character (/).

The slash character (/) and the exclamation mark (!) are not permitted in any topic names.

The following characters are not permitted in topic names for topics that are accessed by publishers or any clients that use the Classic API:

Table 7: Restricted characters

Character	Reason for restriction
/	This is a name separator and cannot occur in a topic name.
[]\^$.|?*+()	These are all metacharacters used in regular expressions. Any topic String that contains any of these characters is assumed to be a topic selector. These characters cannot be used in topic names.

Character	Reason for restriction
Control/Reserved	No characters with a hexadecimal value of less than 0x2D. This includes some punctuation characters such as comma (,).
Whitespace	No characters defined as whitespace in Java (as indicated by the isWhiteSpace method of the Java Character class).

Reserved spaces

The Diffusion branch of the topic tree is reserved for internal use.

Recommendations

Although all Unicode characters (other than the restricted ones mentioned above) are supported to allow for language variations it is highly recommended that only alphanumeric characters are ever used in topic names. Hyphen (-) or underscore (_) can be used as break characters.

Topics

Consider the types of topic you want to use and how. You can also consider the attributes that topics have. The attributes a topic can have can change depending on the topic type.

Topics can be divided into the following categories:

Publishing topics

You can publish data to these topics and the data is distributed to subscribing clients.

The following types of topic are publishing topics:

- Single value
- Record
- Stateless
- Delegated (Deprecated)
- Protocol buffer
- Custom

Paged topics

You can transmit data through these topics, but paged topics do not use the same pub-sub mechanism as publishing topics.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

The following types of topic are paged topics:

- Paged string
- Paged record

Functional topics

You cannot publish data to these topics. Instead, functional topics provide other capabilities to subscribing clients, for example notifications.

The following types of topic are functional topics:

- Slave
- Routing
- Child list
- Service
- Topic notify

Single value topics

A topic that maintains state at the server as a single value.

The value of the topic state is stored as a string. However, the type of the single value can be constrained in certain ways, for example, to hold a string, an integer, or a decimal number. The type of the single value is described using field metadata in the schema. For more information, see Metadata on page 61.

Why use a single value topic?

For the majority of use cases, single value topics are the most appropriate model for the your data. Single value topics are easy to create and update, and the data the topic contains is simpler.

If a structure is required for your data, you can use the design of your topic tree to define the structure.

Single value topics are supported by all client APIs.

By defining the type of the single value of the topic, you benefit from automatic validation and formatting of the data.

Single value topics have their state hosted on the Diffusion server. Compared to delegated topics, which have their state managed by a client, there are fewer communication steps in retrieving the topic state.

Considerations when using a single value topic

A single value topic can only hold textual data. This type of topic data cannot be used to publish non-textual data, such as a PNG or PDF file.

Single value topics cannot be used to make multiple updates transactionally. If you have multiple items of data that you want to publish at the same time and have received by subscribing clients at the same time, you cannot split these items of data across multiple single value topics. In this case, it is more appropriate to use a record topic whose topic data structure contains fields for each of the data items.

Record topics

A topic that maintains state at the server in Diffusion record format. Record format comprises strings separated by field or record delimiters or both.

Describe the layout of the data by using content metadata in the schema. For more information, see Metadata on page 61.

Why use a record topic?

A record topic enables multiple fields to be maintained in the same topic. All updates made at the same time to fields on a record topic are sent out to the client together. This enables a set of fields to be treated as a transactional group.

Deltas of change are calculated at the server such that only those fields that have changed since the last update are sent out to the subscribed clients. This ensures that the minimum amount of data is sent to clients.

If your data structure is too complex to be represented by a topic tree or might make the topic tree structure difficult to manage, it might be more appropriate to represent part of the data structure inside a record topic by using metadata records and fields.

One example where this is the case is when your data includes a variable number of fields of a certain type. If you use single value topics for each of these fields, you must create a topic for the maximum number of occurrences of that field. However, in a record topic, you can define the field at the end of the content and mark it as having variable multiplicity. The record topic can include as many of that field as are required.

Record topics have their state hosted on the Diffusion server. Compared to delegated topics, which have their state managed by a client, there are fewer communication steps in retrieving the topic state.

Considerations when using a record topic

You must define the metadata when you create the record topic. This is more complex than creating a single value topic.

Record topic updates represent all possible records and fields within the content. Fields that have not changed are sent within delta updates as zero-length strings. Because unchanged fields are represented this way a client cannot differentiate between a field that has not changed and an empty field. You can specify a special character that is used to represent an empty field.

DEPRECATED: Delegated topics

A topic that has no data held by the Diffusion server. Its state is delegated to a client.

When a client subscribes or fetches, a client that is registered to provide the state is invoked to retrieve the state. When the client that provides the state updates a delegated topic nothing is stored at the server and the update is fanned out to all subscribed clients.

Why use a delegated topic?

All handling of the topic and the topic state of a delegated topic is done by your client application. Because of this, the format of the data published on a delegated topic is entirely flexible. The topic content is treated as byte data by the Diffusion server. How that byte data is handled and interpreted is determined by your client applications.

Considerations when using a delegated topic

You must write all of the logic in your client to handle the byte data that is published on a delegated topic. This might mean it takes longer to get started using a delegated topic compared to topics that are handled by the Diffusion server.

You also lose some of the benefits of having a topic whose content is understood by the Diffusion server, such as validation, formatting, conflation, and deltas.

When a client fetches the topic state of a delegated topic, the request and response must travel further to get to the client that provides the topic state than are required when the topic state is hosted on the server.

Stateless topics

A topic that has no state held on the Diffusion server or on the client that publishes to it.

A stateless topic has no state. It can be used for publishing and receiving updates, but not for fetching the current topic state.

Stateless topics are the only type of topic provided in other typical pub-sub solutions.

> **Note:** When a topic is created with a hierarchic name, if the intermediate nodes in the tree do not already exist they are automatically created as stateless topics.

Why use a stateless topic?

You can use stateless topics for data streams where there is no current state of the data, only updates. For example, a feed of news items.

All handling of the topic and topic data of a stateless topic is done by your client application. Because of this, the format of the data published on a stateless topic is entirely flexible. The topic content is treated as byte data by the Diffusion server. How that byte data is handled and interpreted is determined by your client applications.

Stateless topics are created automatically at empty intermediate nodes of the topic tree, but you can promote a stateless topic to a different topic type later.

Considerations when using a stateless topic

A stateless topic does not store state. You cannot fetch the topic state and when you first subscribe to a stateless topic you do not receive the topic state as a snapshot as you do with a stateful topic.

You must write all of the logic in your client to handle the byte data that is published on a stateless topic. This might mean it takes longer to get started using a stateless topic compared to topics that are handled by the Diffusion server.

You also lose some of the benefits of having a topic whose content is understood by the Diffusion server, such as validation, formatting, conflation, and deltas.

Routing topics

A special type of topic, which can map to a different real topic for every client that subscribes to it. In this way, different clients can see different values for what is effectively the same topic from the client point of view.

When a client subscribes to a routing topic, the request is either passed to a client that has registered as a routing subscription handler for the topic or handled by a server-side routing handler. The routing handler assigns a linked topic to represent it to that client.

The routing handler can assign a different linked topic to each client that subscribes to the routing topic. The linked topic can be a topic of one of the following types:

- Single value
- Record
- Slave
- Protocol buffer
- Custom

When updates are received on the linked topic, those updates are propagated through the routing topic to the subscribing clients.

The subscribing client is not aware of the linked topic. It is subscribed to the routing topic and all the updates that the client receives contain only the routing topic path and not the linked topic path.

Why use a routing topic?

Use routing topics when you want your subscribing clients to all have the same subscription behavior, but the data they receive to be decided by a routing handler depending on criteria about that client.

For example, your subscribing clients can subscribe to a routing topic called Price, but the routing handler assigns each client a different linked topic depending on the client's geographic location. This way, clients in different countries can act in the same way, but receive localized information.

Considerations when using a routing topic

Using routing topics requires that you write a routing handler that is either hosted on the server or registered by a client with the required permissions.

A subscribing client only needs permission to subscribe to the routing topic. Permission to subscribe to the linked topic is not required.

If the linked topic is removed, subscribing clients are automatically unsubscribed from the routing topic.

Slave topics

A special type of topic that has no state of its own but is a reference to the state of another topic.

A slave topic acts as an alias to another topic, the master topic. Updates published to the master are fanned out to subscribers of the slave. The slave cannot be updated directly. The master topic must be one of the following types of topic:

- Single value
- Record
- Protocol buffer
- Custom

The link between a slave topic and a master topic is defined when the slave topic is created. This is different to routing topics where the link between topics is defined when a client subscribes.

Why use a slave topic?

You can use slave topics to provide the same data from multiple topic paths and manage the topics from only one topic.

You can use a slave topic to act as a redirection to a succession of master topics. For example, you can create a slave topic called latest that is linked to a master topic where data is published about a current event. When that event is no longer current, you can remove the slave topic and recreate it now linked to the master topic where data is published about what is now the current event.

The subscribing clients can subscribe to the latest slave topic and they continue to be subscribed to the slave topic and receive the latest data, even as the master topic that provides the data changes.

Considerations when using a slave topic

A client only needs permissions on the slave topic. Permission to subscribe to the linked topic is not required.

More than one slave can point to the same master topic.

A slave topic cannot also act as a master topic to another slave topic.

Removing a master topic causes all linked slave topics to be removed.

Custom topics

A topic that has its state maintained at the server by a user-written Diffusion class.

By implementing publishing topic data, you can define the types, formatting, and structure data on a custom topic. By writing a custom topic handler you can define how that data is maintained, compared, and sent to subscribing clients.

These user-written classes must be on the classpath of the Diffusion server.

The topic state is delegated to an instance of the class and is handled on the server side. Updates from a client are passed to the custom topic handler for processing. The custom topic handler can hold the topic state internally or elsewhere.

Why use a custom topic?

Custom topics enable you to use data types and layouts that are not supported directly by Diffusion.

Considerations when using a custom topic

The custom topic handler must be present on the classpath of the Diffusion server. If your solution uses multiple Diffusion servers that all host the same custom topic, the same custom topic handler must be present on all of their classpaths.

Custom topic handlers can only be implemented in Java.

Protocol buffer topics

A topic that maintains state at the server in Google protocol buffers format.

Protocol buffers provide an extensible mechanism for serializing structured data.

For more information about protocol buffers, see https://developers.google.com/protocol-buffers/.

Define the format of data on a protocol buffer topic by using a `.proto` file. Compile the `.proto` file into a Java class and ensure that the class is on the Diffusion server classpath.

The schema defines a compiled proto class which must exist at the server and the name of a message definition within the class that defines the topic data layout.

Why use a protocol buffer topic?

Data on protocol buffers topics is simple and small, and can be transmitted and parsed quickly.

A protocol buffer topic benefits from the same delta-processing capability at the server as a record topic.

Serialization and deserialization code can be automatically generated for the data on the topic.

Considerations when using a protocol buffer topic

The compiled proto Java class must be present on the classpath of the Diffusion server. If your solution uses multiple Diffusion servers that all host a protocol buffer topic with the same message definition, the same compiled proto class must be present on all of their classpaths.

Protocol buffer definitions must be compiled into Java.

The supported version of protoc is 2.6.1.

Paged string topics

A topic that maintains server-side state as a number of lines of string data. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Why use a paged string topic?

Unlike most publishing topics, paged topics are not designed to distribute streaming data. Paged topics store tabular data. If your data is of a more tabular form - for example, news items - a paged topic might be most appropriate.

A paged string topic is suitable for simple items of data that can be represented as text.

The state of publishing topics is constantly changing, with no capability to look back at its previous values. With a paged topic you can store all updates on the topic and a client can view the history of previous values, without the client having to store them.

A paged topic can serve different parts of its state to different clients. Any subscribed client can view any page of the data that is required.

If the order of your data is important, a paged record topic ensures that updates are added in the appropriate position. This removes the requirement for a client to have to order the stored data before using or displaying it.

Considerations when using a paged string topic

Viewing paged topics is supported only by the Classic API.

You must be subscribed to a paged topic to open a view on it.

The lines of a paged topic are UTF-8 encoded.

You cannot use topic replication to replicate paged topics between Diffusion servers.

Paged topic data is not supported by the Introspector or the console.

Related Links

Paged topic data in JavaScript on page 512
The JavaScript API provides an interface for using paged topics.

Paged record topics

A topic that maintains server-side state as a number of lines of record data. The schema defines the record metadata that defines the lines. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Why use a paged record topic?

Unlike most publishing topics, paged topics are not designed to distribute streaming data. Paged topics store tabular data. If your data is of a more tabular form - for example, news items - a paged topic might be most appropriate.

A paged record topic, enables you to use metadata to define the format of each line of data. If each line contains multiple items of data, paged record topics enables you structure this data, where a paged string topic does not.

The state of publishing topics is constantly changing, with no capability to look back at its previous values. With a paged topic you can store all updates on the topic and a client can view the history of previous values, without the client having to store them.

A paged topic can serve different parts of its state to different clients. Any subscribed client can view any page of the data that is required.

If the order of your data is important, a paged record topic ensures that updates are added in the appropriate position. This removes the requirement for a client to have to order the stored data before using or displaying it.

Considerations when using a paged record topic

Viewing paged topics is supported only by the Classic API.

You must be subscribed to a paged topic to open a view on it.

The lines of a paged topic are UTF-8 encoded.

You cannot use topic replication to replicate paged topics between Diffusion servers.

Related Links

Paged topic data in JavaScript on page 512
The JavaScript API provides an interface for using paged topics.

Topic notify topics

A functional topic that can be used by clients to receive notifications whenever topics are created, modified, or removed in a part of the topic tree.

A client can subscribe to a topic notify topic to request notifications that contain one of the following levels of notification when a topic is added:

- topic name and type
- topic name, type, and properties
- topic name, type, and metadata
- full topic definition (including name, type, properties, and metadata)

The client uses a handler to handle topic notifications. To receive notifications, the handler must select which parts of the topic tree to receive notifications on. You can use topic names or topic selectors or both to define these parts of the topic tree.

Why use a topic notify topic?

Topic notify topics are useful in a *fan-out solution*, where publisher clients must be notified of the creation of topics on a remote server to in turn create them on their local server.

Considerations when using a topic notify topic

In some APIs, a client that subscribes to a topic notify topic must provide the handler that handles the notifications.

Service topics

A functional topic that implements a request/response type service. The service is implemented as a user-written server-side Diffusion class.

Child list topics

A functional topic that maintains a list of child topics and notifies subscribed clients when a child topic is added or removed directly.

A child topic is a topic that is directly beneath the child list topic in the topic tree. For example, if the child list topic is at the path foo, the topics foo/bar and foo/fred are its child topics. However, the topic foo/bar/baz is not a child topic of foo.

Why use a child list topic?

You can use child list topics to group sets of related data items. Each data item is a child topic beneath the child list topic. A client can subscribe to the child list topic and receive a notification when data item is added to a set (that is, when a child topic is added or removed).

Considerations when using a child list topic

Child list topics are supported only by the Classic API.

Topic attributes

The attributes of a topic can be supplied when it is created, but in most cases the default values for the attributes are what is required.

There are a common set of attributes that apply to all topic types but certain types have additional attributes, some of which are mandatory. The API specifications of the relevant topic details builders indicate what is required.

The following topic attributes are common to all topic types:

Tidy on unsubscribe

By default, if a client unsubscribes from a topic, it continues to receive any updates to that topic that were queued for it before it unsubscribed. By choosing this option, when a client unsubscribes from the topic any updates queued for that topic are removed from the client's queue.

Reference

You can associate a string reference with a topic which is used in logging and monitoring to make the topic easier to identify.

Properties

DEPRECATED: An optional map of properties can be supplied in order to specify topic attributes not supported by the Unified API. This capability is present for backwards compatibility. The key values of the properties are the same as the names in the `TopicProperty` enum in the Java Classic API and the values are string representations of the property value. This cannot be used to set properties that are represented in the Unified API.

Data format

Consider the following aspects of the data you publish to topics or send in messages.

Metadata

Depending on the type of topic you are publishing the data through, you can use metadata to describe the format of the data.

For more information, see Metadata on page 61.

Byte encoding

You can apply a byte encoding to your data that compresses or encrypts the data.

For more information, see Byte encoding of content on page 63.

Character encoding

UTF-8 is the default character set used by Diffusion.

> **Note:** For backwards compatibility with older clients, both the Classic API and the Unified API enable you to set the charset of your strings. However, be aware that if you do this you must set the charset the same on every client and that some APIs do not support all charsets.

UTF-8 can encode any Unicode character but only some of those (for example, ASCII characters) can be held as a single byte. Other characters consume between 2 and 4 bytes. Consider this when sizing for messages that might hold any character data outside the normal ASCII range.

Size

The default maximum size for updates and messages is 32 kilobytes.

In general smaller updates are more efficient. If your updates are large, consider whether the information in them can be split up across multiple topics. For example, if you have updates to a Weather topic that contain temperature, rainfall, cloud cover, and windspeed, it might be preferable to publish the individual pieces of information of separate topics: windspeed on Weather/Windspeed, and so on.

Metadata

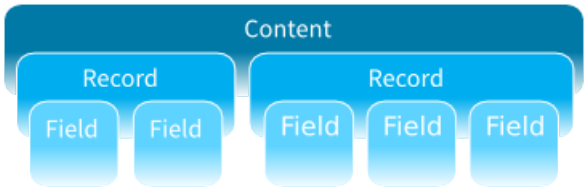
Metadata defines how data is formatted when it is published on a topic. Define the metadata structure for a topic that describes the grouping, order, type, and multiplicity of data items published on a topic.

Updates and messages contain byte data. This byte data can be formatted in whatever way your application requires. For example,

- When creating a record topic, define a metadata structure that describes the data format both for updates published on that topic and those sent on to the subscribing clients.
- When creating a single value topic, you can define field metadata that constrains the data type that updates published to the topic can have.
- When sending a message through a topic path, you can use metadata to create the content of your message.

Metadata structure

The metadata structure is made up of nested records and fields. The outer container is the *content*. This contains one or more *records*. Each record can contain one or many *fields*.



Fields and records are identified by a name. Every record must have a name that is unique within the content. Every field must have a name that is unique within the enclosing record.

Every field or record defined in the metadata structure can represent one or more possible occurrences of that field or record in the byte data. The number of possible occurrences of a record or field is described by its multiplicity.

The order in which records and fields are defined within their enclosing container defines the order that they appear in byte data.

Field metadata

A metadata field defines an elementary data item within a record.

Every field has the following properties:

- Data type
- Multiplicity
- Default value

Data type

The data type of a field defines its actual representation within the byte data. The following table describes the data types that are available.

Table 8: Data types for metadata fields

Data type	Description	Default
String	A character string.	Zero-length string
Integer string	An integer represented in the content as a character string. If a field is defined as this type, it can only contain numeric digits with an optional leading sign. Fields of this type cannot be empty.	0
Decimal string	A decimal number represented in the content as a character string. Decimal fields have the number of places to the right of the decimal point defined by the scale, the default being 2. Such values can be parsed from a character string with any number of digits to the right of the decimal point. Half-up rounding is applied to achieve the target scale. Output of the field is rendered with the specified scale. Fields of this type cannot be empty. For comparison purposes the scale is ignored: a value of 1.50 is the same as 1.5.	0.00 (depending on scale)
Custom string	This is a special type where the behavior is delegated to a user-written custom field handler. This type is available in all topic data types.	-

Multiplicity

The multiplicity of a metadata field or record defines the number of times the corresponding byte data can occur within the enclosing record or content.

Multiplicity is defined in terms of the minimum and maximum number of occurrences. Some byte data representations support variable numbers of records and field, whereas others (such a record data) only support fixed number of records and fields (where minimum=maximum) except in the last position.

Fixed multiplicity is defined by a single number. For example, a multiplicity of 5 on a field indicates that there must be exactly five occurrences of the field within its enclosing record.

Variable multiplicity is defined by a minimum value and a maximum value and is represented with the notation *n*..*n*. For example, multiplicity of 1..5 on a field specifies that there must be between one and five occurrences of the field within its enclosing record.

A special maximum value of -1 is used to represent no maximum. For example, a multiplicity of 1..-1 on a field specifies there can be any number of occurrences of the field, but there must be at least one.

Optional nodes are defined by a minimum value of 0. For example, a multiplicity of 0..1 on a field specifies that there can be zero of one occurrences of the field within its enclosing record. A fixed multiplicity of 0 is not allowed.

Variable multiplicity fields must be defined at the end of their containing record. Variable multiplicity records must be defined at the end of the content.

Default value

You can specify a default value for a field. If you do not specify a default value, the default value for the data type is used. When content is created using metadata, default initialization applies the default values specified for each field.

Byte encoding of content

Diffusion provides a number of byte encodings that can be applied to the data when it is sent over a connection.

Only one byte encoding can be specified for your content. If you specify an encoding for the content then specify another encoding, only the second encoding that you specified is applied.

The encodings are listed in the following table:

Table 9: Types of byte encoding

Compressed	The data is compressed using the ZLIB compression library. This compression is effective in compressing sections of text.
Encrypted	The data is encrypted using an algorithm based on XXTEA.
Base64	The data is encoded in base-64 notation.

Diffusion handles the encoding and decoding (or compression and decompression) of data. Clients are not required to handle encoding and decoding (or compression and decompression).

Because the Diffusion server handles the encoding and decoding (or compression and decompression), the process is transparent to the client.

Note: Some APIs do not support all of these encodings. For more information, see the API documentation.

Transport capabilities

When a client connects to the Diffusion server, the server establishes whether or not the connection has the capability to support the encodings. If the connection can support the encoding that is specified for a message, the Diffusion server applies that encoding to the message. If the connection does not support the encoding, the message is transmitted unencoded.

The following table lists the encodings supported by each transport:

Table 10: Encoding support transports

Transport	Compressed	Encrypted	Base64
DPT	YES	YES	YES
WebSocket	NO	NO	NO
HTTP Duplex	YES	YES	YES
HTTP Comet	YES	YES	YES
HTTP	NO	NO	NO
iFrame	NO	NO	NO

Publication

Having decided on your topic structure and the format of your data, consider how you publish the data through the topics.

Pub-sub is the primary model of data distribution used by Diffusion. Clients subscribe to a topic. When data is published to the topic as an update, the Diffusion server pushes that update out to all of the subscribed clients.

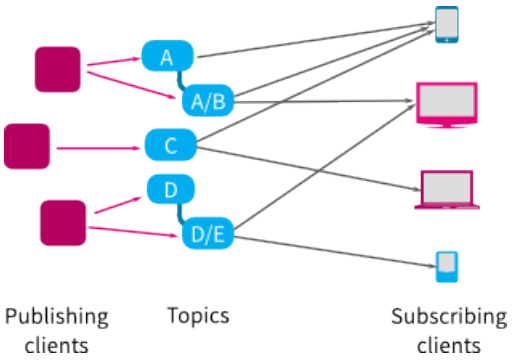


Figure 6: Pub-sub model

A client can both publish to topics and subscribe to topics, depending on the permissions that client has.

Concepts

Update

An update is data published to a topic by a client or publisher that is applied to the topic to change the topic state. The updated data is then pushed out to all subscribing clients.

State

The latest published values of all data items on the topic. The state of a topic is stored on the Diffusion server.

Snapshot

A snapshot is an update that contains the current state of all data on the topic.

Delta

A delta is an update that contains only those items of data that have changed on the topic since the last update was sent to the client. Topics that contain only a single item of data cannot use delta messages.

Topic loading

When a client first subscribes to a topic, it is sent a topic load message. A topic load is a snapshot update that contains the current state of the topic.

Fetch

A request for the current state of all data on the topic. A client can fetch a topic's state without being subscribed to the topic. This request-response mechanism of getting data from a topic is separate from the pub-sub mechanism.

Publishing data

Consider the following information when deciding how to publish data to topics.

Data format

The updates that you publish to a topic must have a data format that matches the data format of the topic.

For example, if your topic is a single value topic where the data is of type integer, all updates published to the topic must contain a single piece of integer data.

Similarly, if your topic is a record topic with a metadata structure defined, all updates published to the topic must have the same metadata structure.

Dynamically adding topics

A publishing client can create topics dynamically as and when the topics are required. For example, in response to a subscription request from another client for a non-existent topic.

Security

To publish data to a topic, a client must have the update_topic permission for that topic.

For more information, see Permissions on page 138.

Subscribing to topics

Consider the following information when deciding how clients subscribe to topics.

Subscribing to multiple topics

A client can subscribe to multiple topics in a single subscribe request. This subscription can be to topics that match a particular regular expression or to topics in a particular branch of the topic tree.

Use topic selectors to define a set of topics.

For more information, see Topic selectors in the Unified API on page 66 and Topic selectors in the Classic API on page 72

Subscribing to topics that do not exist

A client can subscribe to a topic that does not exist. This is pre-emptive subscription. Diffusion keeps track of all subscribe and unsubscribe requests from a client, including for topics that do not exist. When a new topic is created, if a client has subscribed to that topic path, the client is subscribed to the new topic and receives updates from it.

Forced subscription

A publisher or a client with the required permissions can force other clients to become subscribed to a topic or set of topics, even if the subscribing client did not request the subscription.

Security

To subscribe to a topic, a client must have the read_topic permission for that topic.

For more information, see Permissions on page 138.

Topic selectors in the Unified API

A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, C Unified API

Pattern expressions

Use pattern expressions to create a topic selector of one of the types described in the following table. The type of the topic selector is indicated by the first character of the pattern expression.

Table 11: Types of topic selector

Topic selector type	Initial character	Description
Path	>	A path pattern expression must contain a valid topic path. A valid topic path comprises topic names separated by path separators (/). A topic name comprises one or more UTF characters except for slash (/). A PATH selector returns only the topic with the given path. See Path examples on page 68

Topic selector type	Initial character	Description
Abbreviated path	Any character except the following: • Hash symbol (#) • Question mark (?) • Greater than symbol (>) • Asterisk (*) • Dollar sign ($) • Percentage symbol (%) • Ampersand (&) • Less than symbol (<)	An abbreviated path pattern expression is an alternative syntax for a path pattern selector. It must be a valid topic path. A valid topic path comprises topic names separated by path separators (/). A topic name comprises one or more UTF characters except for slash (/). Abbreviated path pattern expressions are converted into PATHselectors and return only the topic with the given path. See Abbreviated path examples on page 69
Split-path	?	A split-path pattern expression contains a list of regular expressions separated by the / character. Each regular expression describes a level in the topic hierarchy. A SPLIT_PATH_PATTERN selector returns topics for which each regular expression matches the part of the topic path at the corresponding level. See Split-path examples on page 70
Full-path	*	A full-path pattern expression contains a regular expression. A FULL_PATH_PATTERN selector returns topics for which the regular expression matches the full topic path. See Full-path examples on page 70 **Note:** Full-path pattern topic selectors are more powerful than split-path pattern topic selectors, but are evaluated less efficiently at the server. If you are combining expressions, use selector sets instead.
Selector set	#	A selector set pattern expression contains a list of selectors separated by the separator ////. A SELECTOR_SET topic selector returns topics that match any of the selectors. **Note:** Use the `anyOf()` method for a simpler method of constructing SELECTOR_SET topic selectors. See Selector set examples on page 71

Diffusion topic selectors use Java-style regular expressions. Any Java-style regular expression can be used in split-path and full-path patterns, with the following restrictions:

- It cannot be empty
- In split-path patterns, it cannot contain the path separator (/)
- In full-path patterns, it cannot contain the selector set separator (////)

For more information about Java-style regular expressions, see Java regular expressions.

Descendant pattern qualifiers

You can modify split-path or full-path pattern expressions by appending a descendant pattern qualifier. These are described in the following table:

Table 12: Descendant pattern qualifiers

Qualifier	Behavior
None	Select only those topics that match the selector.
/	Select only the descendants of the matching topics and exclude the matching topics.
//	Select both the matching topics and their descendants.

Topic path prefixes

The topic selector capabilities in the Unified API provide methods that enable you to get the topic path prefix from a topic selector.

A topic path prefix is a concrete topic path to the most specific part of the topic tree that contains all topics that the selector can specify. For example, for the topic selector `?foo/bar/baz/.*/bing`, the topic path prefix is `foo/bar/baz`.

The topic path prefix of a selector set is the topic path prefix that is common to all topic selectors in the selector set.

Path examples

The following table contains examples of path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
>alpha/beta	yes	no	
>/alpha/beta/	yes	no	This pattern expression is equivalent to the pattern expression in the preceding row. In an absolute topic path, single leading or trailing slash characters (/) are removed because the topic path is converted to canonical form. A path pattern expression can return a maximum of one topic. The trailing slash in this example

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
			is not treated as a descendant qualifier and is removed.
>alpha/beta/gamma	no	yes	
>beta	no	no	The full topic path must be specified for a path pattern expression to match a topic.
>.*/.*	no	no	For clients using the Unified API, the period (.) and asterisk (*) characters are valid in topic names. In a path pattern expression these characters match themselves and are not evaluated as part of a regular expression.

Abbreviated path examples

The following table contains examples of abbreviated path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
alpha/beta	yes	no	
/alpha/beta/	yes	no	This pattern expression is equivalent to the pattern expression in the preceding row. In an absolute topic path, single leading and trailing slash characters (/) are removed because the topic path is converted to canonical form. A path pattern expression can return a maximum of one topic. The trailing slash in this example

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
			is not treated as a descendant qualifier and is removed.
alpha/beta/gamma	no	yes	
beta	no	no	The full topic path must be specified for a path pattern expression to match a topic.

Split-path examples

The following table contains examples of split-path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
?alpha/beta	yes	no	
?alpha/beta/	no	yes	The trailing slash character (/) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns descendants of the matching topics, but not the matching topics themselves.
?alpha/beta//	yes	yes	Two trailing slash characters (//) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns matching topics and their descendants.
?alpha/beta/gamma	no	yes	
?beta	no	no	
?.*	no	no	Each level of a topic path must have a regular expression specified for it for a split-path pattern expression to match a topic.
?.*/.*	yes	no	
?alpha/.*//	yes	yes	In this pattern expression, “alpha/.*” matches all topics below alpha in the topic hierarchy. The descendant pattern qualifier (//) indicates that the matching topics and their descendants are to be returned.

Full-path examples

The following table contains examples of full-path pattern expressions:

Expression	Matches alpha/beta?	Matches alpha/beta/gamma?	Notes
*alpha/beta	yes	no	
*alpha/beta/gamma	no	yes	
*alpha/beta/	no	yes	The trailing slash character (/) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns descendants of the matching topics, but not the matching topics themselves.
*alpha/beta//	yes	yes	Two trailing slash characters (//) is treated as a descendant pattern qualifier in split-path pattern expressions. It returns matching topics and their descendants.
*beta	no	no	In a full-path pattern selector the regular expression must match the full topic path for a topic to be matched.
*.*beta	yes	no	The regular expression matches the whole topic path including topic separators (/).

Selector set examples

Use the `anyOf` methods to create a SELECTOR_SET `TopicSelector` object.

The following example code shows how to use the `anyOf(TopicSelector... selectors)` method to create a selector set topic selector:

```
// Use your session to create a TopicSelectors instance
TopicSelectors selectors = session.topicSelectors();

// Create topic selectors for the individual pattern expressions
TopicSelector pathSelector = selectors.parse(">foo/bar");
TopicSelector splitPathSelector = selectors.parse("?f.*/bar\d
+");
TopicSelector fullPathSelector = selectors.parse("*f.*\d+");

// Use the individual topic selectors to create the selector set
 topic selector
TopicSelector selector = selectors.anyOf(pathSelector,
 splitPathSelector, fullPathSelector);

// Use the topic selector as a parameter to methods that perform
 actions on topics or groups of topics
```

The following example code shows how to use the `anyOf(String... selectors)` method to create the same topic selector as in the previous code example, but in fewer steps:

```
// Use your session to create a TopicSelectors instance
TopicSelectors selectors = session.topicSelectors();
```

```
// Create the selector set topic selector by passing in a list
 of
// pattern expressions
TopicSelector selector = selectors.anyOf(">foo/bar", "?f.*/bar\d
+", "*f.*\d+");

// Use the topic selector as a parameter to methods that perform
 actions on topics or groups of topics
```

Related Links

Topic tree on page 49
Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

Topic selectors in the Classic API on page 72
A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Topic tree on page 49
Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

Topic selectors in the Classic API

A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Supported in: JavaScript Classic API, Java Classic API, .NET Classic API, C Classic API, iOS Classic API, Android Classic API, Flash® Classic API, Silverlight Classic API

Including subordinate topics

When specifying a topic name you can also indicate that all of its subordinate topics are to be included by suffixing the name with a slash character (/).

For example, to select all of the subordinate topics of a topic named MyTopic use a selector of the format `MyTopic/`.

This notation can also be used with hierarchic topic names. So to select all topics subordinate to the topic named A/B, use a selector of the following format:

`A/B/`

Specifying a suffix of "/" does not include the topic named prior to the final "/". To include the specified topic and all of its subordinates, use "//".

For example, to select the topic A/B and all of its subordinates, specify the following selector:

`A/B//`

Fuzzy matching

A number of topics can be selected using a single topic selector which uses regular expressions to match against topic names.

Regular expressions provide a powerful mechanism for String pattern matching which is not discussed here. A Java tutorial is available for those familiar with Java or a more generic tutorial can be consulted for other language users. The regular expression syntax supported by Diffusion is defined by the Java Pattern class.

The important point to note about the use of regular expressions in topic selectors is that they are hierarchic (separated by /) and that each part of a selector can be a regular expression pattern. A multi-part selector is evaluated part by part starting from the top level of the topic tree in an attempt to find a matching topic in the tree.

If there is no "/" in the selector string but there are regex characters then the whole string is applied as a regex against the full topic name. This can be useful in some circumstances, for example when you do not know how many topic levels exist but it is far less efficient than per-level matching.

Example of selector regular expression processing

Consider the following topic selector:

```
A.*/B/.*X
```

As the above selector has three specifications it only matches with topic names with three parts.

The first check is to select any topics at the top-level of the topic tree whose name matches the pattern "A.*". In regular expression notation this pattern matches with any String that starts with "A" and is followed by zero or more other characters. So if there are no top-level topics that start with an "A" then this selector matches with no topics at all. However, for any top-level topic that does match, processing goes to the next part which is a simple String "B" and only matches with a subordinate topic called "B". So if any topics called "B" are found under the top-level topics starting with "A", processing goes onto the final pattern ".*X" which in regex notation indicates any String with any number of characters before a final "X" (a String ending with "X").

So to summarize, the above selector matches only with hierarchical topic names with 3 elements where the first element starts with an "A", the second element is "B" and the third element ends with an "X".

The following are matches:

```
Accounts/B/TAX
```

```
A/B/X
```

```
Admin/B/ProjectX
```

```
But the following are not:
```

```
Accounts/TAX
```

```
Admin/B/ProjectYHR/B/TAX
```

If you do not know how many levels you are dealing with but want to select topics where the lowest level topic name is ABC, you must use a whole topic name selector as follows:

```
.*\x2FABC
```

The use of "\x2F" is necessary to represent a "/" as otherwise the selector evaluates per level.

So this selector matches with:

```
A/B/ABCX/Y/ZZZ/wwwwww/ABC
```

but not with:

```
A/B/CABC
```

Mixed Mode selectors

It is permitted to mix regex pattern handling and subordinate topic suffixes in the same topic selector pattern.

So it is permitted to use a selector of the form A.*/Address/ which has the effect of selecting all of the topics subordinate to the topic named Address within any top level topic starting with "A".

Selector examples

Table 13: Selector examples

Selector pattern	Selects	Examples
A/	All topics subordinate to, but not including the top-level topic named "A".	A/B, A/B/C
A//	The top level topic named "A" and all topics subordinate to it.	A, A/B, A/B/C
A.*	All topics that start with the letter "A"	A, Accounts, Admin
A.*//	All topics that start with the letter "A" and their subordinates.	A, Accounts, A/B, Admin/X/Y
Counties/.*ex	All topics under the top-level topic called Counties whose name ends in "ex".	Counties/Middlesex, Counties/Sussex
.*folk	All topics whose full name ends in "folk" regardless of level.	UK/Counties/Suffolk, Counties/Norfolk

When is a topic string a selector?

In certain parts of the APIs only topic names can be specified (for example, when adding a topic) but in other areas selectors are allowed (for example, in subscription).

A topic string is considered to be a selector if it is terminated by a "/" or it contains any one of the regular expression metacharacters which are defined as the following characters: `[]\^$.|?*+()`

Related Links

Topic tree on page 49
Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

Topic selectors in the Unified API on page 66
A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topic tree on page 49

Diffusion primarily distributes data using a pub-sub model, where content is published to topics. These topics are arranged as a tree.

Distributing and viewing data as pages

Paged topics act differently to standard publishing topics. Distributing data through paged topics

Concepts

Page

A page is a subset of the tabular data stored on a paged topic. Each page is made up of a certain number of lines.

Line

A line is a single item of data stored on a paged topic. A line can be a string or a record.

Add

Lines are added to a paged topic.

Update

The content of lines can be updated.

Remove

Lines can be removed from a paged topic.

View

A view is a representation of data on a paged topic that can be opened by a subscribing client. The subscribing client can open a view on the paged topic that includes a specified number of lines of the topic data.

Ordering

Lines on a paged topic can be ordered according to simple or user-defined criteria.

Dirty

If a line is added into a paged topic in a position that is open in a subscribing client's view. That view is dirty and must be refreshed to show the latest topic data.

Behavior

Paged topics have state which is stored in a tabular format, as pages of lines. Unlike publishing topics, clients do not publish updates to a paged topic to change the state. Instead clients add, update, or remove lines in the topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic. Lines that are added to unordered paged topics are added at the end. Lines that are added to ordered paged topics are added at the position defined by the comparator or rules.

A client must subscribe to a paged topic to be able to access the data on it. However, unlike publishing topics clients do not receive updates whenever changes are made to the paged topic state. To access the data, the client must open a view on the paged topic and specify how many

lines per page and what page to open the view on. The client can then page through the data. If the state of the client's current page changes, the client is notified and can choose to refresh the page.

Conflation

Conflation of messages is the facility to treat two messages as being essentially the same and avoiding sending duplicate information to clients.

Updates and messages for a client are queued on the Diffusion server. Each client has its own outbound queue.

Conflation removes an existing message from the outbound client queue and replaces it with a newer, equivalent message either at the position of the old message or at the end of the client queue. The replacement message can either be a new update or a merge of the new update and an existing update.

Conflation is an optional feature that can be applied to all clients, clients connecting through a specific connector, or can be applied programmatically on a client-by-client basis.

Advantages of message conflation

There are many scenarios in real-time data distribution where data being communicated need not only be current, but must always be consistent. Structural conflation maximizes for concurrency while ensuring consistent delivery.

Concurrency

Both simple and structural conflation maximize for concurrency through avoiding distributing soon to be stale data. The higher the rate of change, the higher the value extracted. On other words, where clients or servers are running near to saturation based on available connectivity Diffusion can automatically adapt to this load by minimizing the data distributed.

In addition to the load-adaptive nature of conflation the effect is fair for clients connected to the same topic. The frequency of distributed changes is evenly amortized across clients.

Conflation favors currency and reduces (but cannot entirely eliminate) the delivery of stale data.

Consistency

Structural conflation synthesizes complex event processing techniques in a highly efficient (lock-free wait-free concurrency) form inside the server. The specific knowledge of data structures and the semantic concerns for distributing data for a given topic in a given system allows consistent views of the data to be delivered in a way that is not possible with messaging technologies that treat messages as opaque.

Considerations when using conflation

- Do not use conflation if there are relationships or dependencies between topics. Conflation alters the order of updates. If a conflated topic is temporally or causally related to an other topic, conflation an cause unwanted behavior.
- Do not use conflation if individual updates carry forensic storage or audit trail requirements.
- Delta updates are conflated. Snapshots are not conflated.
- Normal priority updates are conflated. High or low priority updates are not conflated.
- Messages that require acknowledgment are not conflated.

Types of message conflation

The types of conflation are *simple conflation*, where a new update replaces an older update, and *structural conflation*, where the data from two updates is combined in accordance with a user-defined operation to create a new update. Both types of conflation can either append the new update to the end of the queue or replace an existing update on the queue.

No conflation

Figure 7: Message flow without conflation enabled

With no conflation, a stream of messages to a client is delivered to the client in the order that they are published or sent. In example shown in the diagram, two messages for topic A, one message for topic B, and two messages for topic C are ready to send to the client.

This is a scenario common to all messaging platforms.

Simple conflation

When a new message is published or sent to a topic, simple conflation removes any earlier messages for that topic from the queue. The new message can be added to the queue either in the position of the earlier message that was removed (replace) or at the end, preserving the original message order (append).

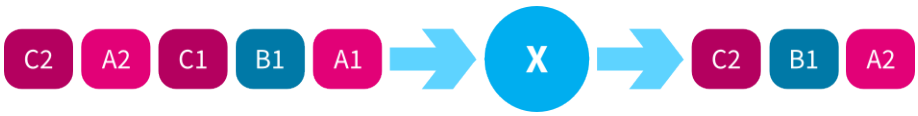


Figure 8: Message flow with simple replace conflation enabled

With simple replace conflation, only the most recent message for a topic is delivered to the client. In the example shown in the diagram, the first message published or sent to topic A is removed and the second message is added to the queue in the position that the first message occupied. The same occurs for the messages sent to topic C.

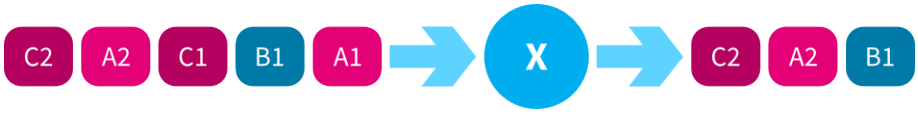


Figure 9: Message flow with simple append conflation enabled

With simple append conflation, only the most recent message for a topic is delivered to the client. Messages are delivered in time order. In the example shown in the diagram, the first message published or sent to topic A is removed and the second message is added to the end of the queue. The same occurs for the messages sent to topic C.

> **Note:** Use this option with care because there is the possibility that messages for a topic can continue to be added to the end of the queue and a value not be delivered for the topic.

In both the append and the replace example, although five messages were ready to send, only three messages were sent. This saves bandwidth and ensures clients receive current data only.

Structural conflation

Structural conflation allows a user-defined operation to be plugged into Diffusion so that rather than refreshing stale data with fresh data, a computation can be performed to merge, aggregate, reverse or combine the effects of multiple changes into a single consistent and current notification to the client.

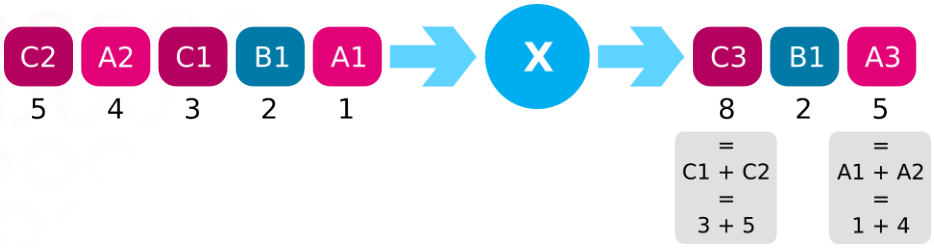


Figure 10: Message flow with merge and replace conflation enabled

In the example shown in the diagram, the operation is the summation of numeric data. The user provides the merge algorithm, that is the summing of the values of two successive messages, and Diffusion sends a single resulting message rather than the individual messages that were combined. The messages A3 and C3 are new messages generated from the merging process.

This is suitable for any scenario where the result is required but individual components that combine to form the result are not required.

The preceding example shows merge and replace. You can merge and append in a similar way as described for simple conflation above.

Various options are available to the user-written merger so that instead of returning a merged message it can indicate that either of the input messages be queued or that the no conflation option be chosen.

Selection of messages for conflation

The preceding examples assume that when a new message is queued for a client it replaces or merges with the last message queued for the message topic. This is the default behavior. When operating conflation in this way there is only ever one message per conflated topic awaiting delivery to the client.

You can specify a user-defined matcher that is used to determine the message that is to be replaced or merged with. This can be used to inspect the content of the messages queued for a topic to select which to conflate. When operating conflation in this way it is likely there can be more than one message per conflated topic awaiting delivery to the client.

How conflation works

Conflation is implemented as a lock-free, wait-free algorithm and can be scaled to meet demands of large numbers of concurrently subscribed clients.

Conflation policies

Conflation policies configure how conflation takes place on topics. One or more conflation policies can be configured, each defining different conflation mechanisms. The policies can define the following behaviors:

- How messages are matched
- Whether replacement is done in place or by appending
- How to merge the two messages

Conflation process

When conflation is enabled for a client, every time a new update is enqueued for the client it is considered for conflation.

1. The Diffusion server checks whether a conflation policy is mapped to the topic that the update is published on.
2. If a policy is mapped to the topic, the matching criteria of the policy is used to scan the updates for that topic that exist on the queue (from newest to oldest) for an existing update that matches the one being queued. If no match is found, the new update is queued at the end of the queue as normal.

   > **Note:**
   >
   > Fast caches and lookup mechanisms are used to find policies and updates in the queue. The whole client queue is not scanned when looking for a match, only updates for the same topic.
   >
   > If default matching (by topic) is used, no comparison with the existing updates is required. This means that the conflation mechanism is as efficient as possible.
3. If the matching criteria finds a matching update in the queue, the conflation policy defines the following behaviors:
   - Whether the update to queue as a result of conflation is the new update or an update produced by merging the content of the matching update and new update.
   - Whether the update to queue replaces the matching update or the matching update is removed and the new update added to the end of the queue.

Conflation occurs on a client-by-client basis in the multiplexer thread. Results of merge actions are cached to ensure that merges of identical message pairs are not repeated for different clients.

Messaging

In addition to publishing data to topics as updates, you can send data as messages to specific clients or to registered message handlers on topic paths.

Messaging and pub-sub are separate capabilities. Messaging uses topic paths to route the messages, but does not use the topics bound to those paths. Sending messages does not change the state of data on the topic, nor does it cause updates to be sent.

There are two ways in which messages are sent through topic paths:

Send messages to a topic path

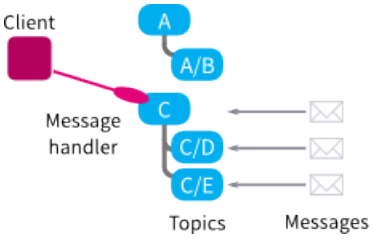


Figure 11: A client registers a handler on part of the topic tree

A client with the send_to_message_handler permission can send messages on a topic path. The sending client does not know which client or publisher, if any, receives the message.

A client with the register_handler permission can register a handler on a part of the topic tree. This client receives any messages that are sent on topic paths in that part of the topic tree.

Send messages to a client session

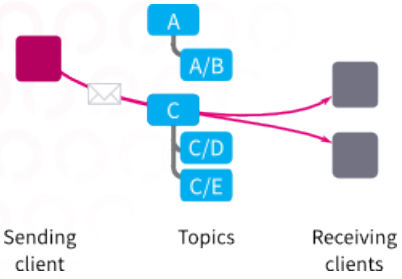


Figure 12: A client can send messages through a topic path to known client sessions

A client with the send_to_session and view_session permissions that knows the session ID of a client session can send messages through a topic path to the known client session.

Considerations when using messaging

- The data format of the content sent in the message is not required to match the data format of the topic, if any, bound to the topic path you use to send the message.
- A message can include headers. These headers are a set of string values that are separate to the message content.
- You can set a priority on messages sent to client sessions. This priority affects the order in which the messages are queued on the Diffusion server for delivery to the client.
- The following APIs require that a topic exists at a topic path to send a message through it and that they are subscribed to a topic path to receive a message through it:
  - Android Classic API
  - Flash Classic API
  - iOS Classic API
  - Silverlight Classic API
  - C Classic API
  - Java Classic API
  - JavaScript Classic API
  - .NET Classic API

Designing your solution

Decide how your solution components interact to most efficiently and securely distribute your data.

There are a number of things to consider when designing your Diffusion solution:

- The number, distribution, and configuration of your Diffusion servers
- The additional components to develop
- How to use load balancers with Diffusion
- Securing your solution

These considerations are not separate. The decisions you make about one aspect of your solution can affect other aspects.

Servers

Consider the quantity, distribution, location and configuration of your Diffusion servers.

How many Diffusion servers?

Consider the following factors when deciding how many Diffusion servers to use in your solution:

Number of client connections

How many client connections do you expect to occur concurrently? For a greater number of concurrent client connections, you might require more Diffusion servers to spread the load between.

Volume of data

At what rate are you publishing updates and sending messages? How large are the updates and messages? If you are distributing a greater volume of data, you might require more Diffusion servers to spread the load between.

Hardware capabilities

The number of concurrent client connections and the volume of data that a single Diffusion server can handle depend on the hardware that the Diffusion server runs on.

In order of importance, the following hardware components have the biggest impact on the server performance:

- Network interface controller (NIC)
- Central processing unit (CPU)
- Random access memory (RAM)

Resilience and failover requirements

Ensure that you have enough Diffusion servers that if one or more becomes unavailable, for example when updating the server or due to a failure of the hosting system, the remaining Diffusion servers can spread the resulting load increase.

You can also use replication between Diffusion servers to increase your solution's resilience. For more information, see Replication on page 82.

Distribution of servers

How you wish to distribute your servers has an effect on how many servers you require.

For example, if your client base is distributed geographically, you might want to locate your Diffusion servers in different territories. This enables your servers to be more responsive because of their proximity to clients. In this case, the number of territories your client base is spread over affects the number of servers you require.

You can easily scale your solution by adding additional Diffusion servers if your requirements change.

How are your Diffusion servers configured?

Consider the following factors when deciding how to configure the Diffusion servers in your solution:

Ports

What ports do you want to provide access to your Diffusion server on? You can configure the Diffusion server to allow the following kinds of connections on individual ports:

- Unified API clients only
- Classic API clients only
- Both Unified API and Classic API clients
- Policy file requests only
- JMX clients

By default, your Diffusion server supports client connections on port 8080.

Reconnection behavior

Do you want to allow clients that lose their connection to reconnect to the server? How long do you want to keep client sessions alive for after the client loses connection?

Not all client types can take advantage of reconnection. For more information, see the section of the manual for each specific API.

Replication

Replication enables Diffusion servers to share information about topics and client sessions with each other through a data grid.

For more information, see Replication on page 82.

Performance

Tuning your Diffusion servers for performance is best done as part of testing your solution before going to production. This enables you to observe the behavior of your solution in action and configure its performance accordingly.

For more information, see Tuning on page 704.

For more information, see Using the configuration API on page 576.

This manual describes the factors that you must consider when designing your Diffusion solution. However, these factors are too many and too interlinked for this manual to provide specific guidance.

Push Technology provides Consulting Services that can work with you to advise on a solution that best fits your requirements.

Replication

Consider how to replicate session and topic information between Diffusion servers to increase availability and reliability.

Diffusion uses a datagrid to share session and topic information between Diffusion servers and provide high availability for clients connecting to load-balanced servers.

Figure 13: Information sharing using a datagrid

Diffusion uses Hazelcast™ as its datagrid. For more information about how Diffusion and Hazelcast interact, see http://www.pushtechnology.com/2015/01/23/replication-pt-3-hazelcast/.

The datagrid is responsible for the formation of clusters and the exchange of replicated data.

Servers reflect session and topic information into the datagrid. If a server becomes unavailable, another server can access the session and topic information that is stored in the datagrid and take over the responsibilities of the first server.

Session replication

You can use session replication to ensure that if a Classic API client connection fails over from one server to another the state of the client session is maintained.

When a connection from a client through the load balancer to a Diffusion server fails, the load balancer routes the client connection to another Diffusion server. This server has access to the session and client information that is replicated in the datagrid.

Clients that connect to a specific Diffusion server and not through a load balancer cannot use session replication.

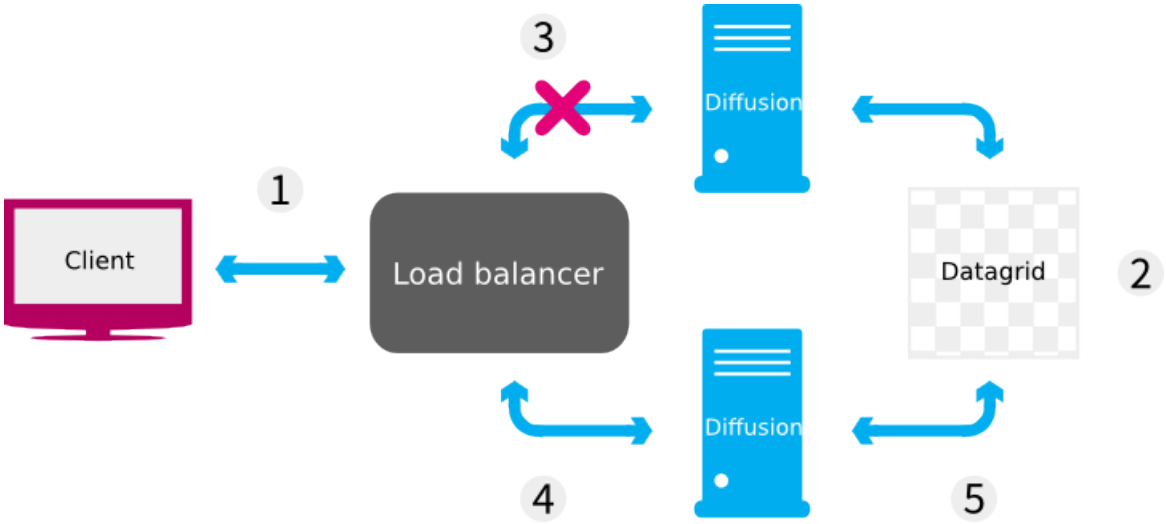


Figure 14: Session replication

1. A client connects to a Diffusion server through a load balancer.

   The load balancer is configured to route based on the client's session ID and requests from the client go to the same server until that server becomes unavailable.

2. Information about the client session is reflected into the datagrid.

   The following information is replicated:

   - session ID
   - session state
   - session principal
   - session properties
   - list of topic selections

   The following information is not replicated and is created anew on the server a client fails over to:

   - session start time
   - statistics
   - client queue
3. A client loses connection to the Diffusion server if the server becomes unavailable.
4. The client can reconnect and the load balancer routes the connection to another Diffusion server.
5. This Diffusion server has access to all of the client information shared into the datagrid by the first Diffusion server.
6. The server uses the list of topic selections to recover the set of subscribed topics and subscribes the client to these topics.
7. Subscribing the client to topics provides snapshot messages for all topics that contain the current topic state.

The client can reconnect to its session only if it reconnects within the reconnect time specified in the `Connectors.xml` configuration file. If the client does not reconnect within that time, the client session information is removed from the datagrid.

Considerations

Consider the following factors when using session replication:

- Replication of session information into the datagrid is not automatic. It must be configured at the server.
- Messages in transit are not preserved. Use acks to ascertain whether or not messages have been received.
- When a client session reconnects it must be authenticated again. Ensure that all Diffusion servers in your solution have access to the same authentication methods.
- The failover appears to the client as a disconnection and subsequent reconnection. To take advantage of high server availability, clients must implement a reconnect process.
- The Diffusion server that a client reconnection attempt is forwarded to depends on your load balancer configuration. Sticky load balancing can be turned on to take advantage of reconnection or turned off to rely on session replication and failover.

Differences between session reconnection and session failover

When a client loses a load balanced connection to Diffusion, one of the following things can occur when the client attempts to reconnect through the load balancer:

Session reconnection

The load balancer forwards the client connection to the Diffusion server it was previously connected to, if that server is still available.

Session failover

The load balancer forwards the client connection to a different Diffusion server that shares information about the client's session through the datagrid, if session replication is enabled.

To a client the process of disconnection and subsequent reconnection is the same for session reconnection and session failover with some slight differences.

Session reconnection	Session failover
The client connects to the same Diffusion server it was previously connected to.	The client connects to a different Diffusion server to the one it was previously connected to.
The client sends its last session ID to the server.	
The server authenticates the client connection.	
The server uses the session ID to lookup the client's message queue and recover it.	The server uses the session ID to retrieve the session state and topic selections from the datagrid.
The server uses the session ID to lookup up the message queue and recovers it, sending any messages that have been queued since the session disconnected.	The server uses the state to recover the session, uses the topic selections to match the subscribed topics, and sends snapshot messages for all the subscribed topics.

Related Links

Replication.xml on page 580
This file specifies the schema for the replication properties.

Client reconnection on page 711

You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep alive the client session.

Topic replication

You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

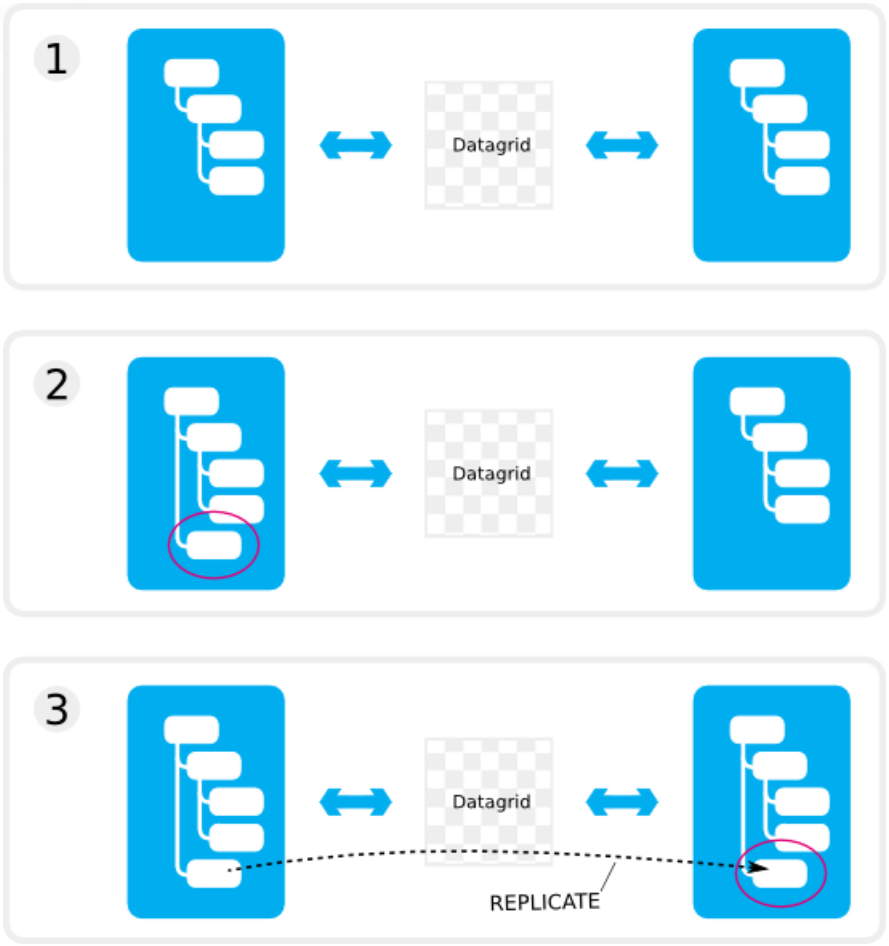


Figure 15: Topic replication

1. Servers with topic replication enabled for a section of the topic tree share information about that section of the topic tree through the datagrid. The topic information and topic data are synchronized on all the servers.
2. A new topic is created on one server in the replicated section of the topic tree.
3. The new topic is replicated on the other servers with identical topic information. When its topic data is updated on the first server, that data is replicated on the other servers.

Considerations

Consider the following factors when using topic replication:

- Only publishing topics can be replicated.
- Replication is supported only for the following types of topic data:
  - No topic data
  - Custom topic data

    - Protocol buffer topic data
    - Record topic data
    - Single value topic data
- Replication is not supported for paged topics.
- Only topic-wide messages are replicated. Messages sent to a single client or to all clients except one are not replicated.
- Replication of topic information into the datagrid is not automatic. It must be configured at the server. This gives a performance advantage, as you can choose which parts of your topic tree to replicate.
- Replication of topic data can impact performance.
- Do not use topic replication on sections of the topic tree that are owned and updated by publishers. Publishers can make updates to topics that are not replicated or that supersede replicated data. If you use topic replication with topics updated by publishers, this can cause the data on the replicated topics to become unsynchronized.
- Avoid including replicated topics in REMOVE_TOPICS session wills. When a replicated topic is removed from a server as a result of a session will, it is removed from all other servers that replicate that topic. If all sessions on a Diffusion server that have a remove topics removal request for a branch of the topic tree close, the topics are removed even if that topic is replicated and sessions on other Diffusion servers have removal requests registered against that part of the tree. When the topics are removed on the server, that change is replicated to all other servers that participate in replication for these topics.

Related Links

Replication.xml on page 580
This file specifies the schema for the replication properties.

Failover of active update sources

You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, another server is assigned to

be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

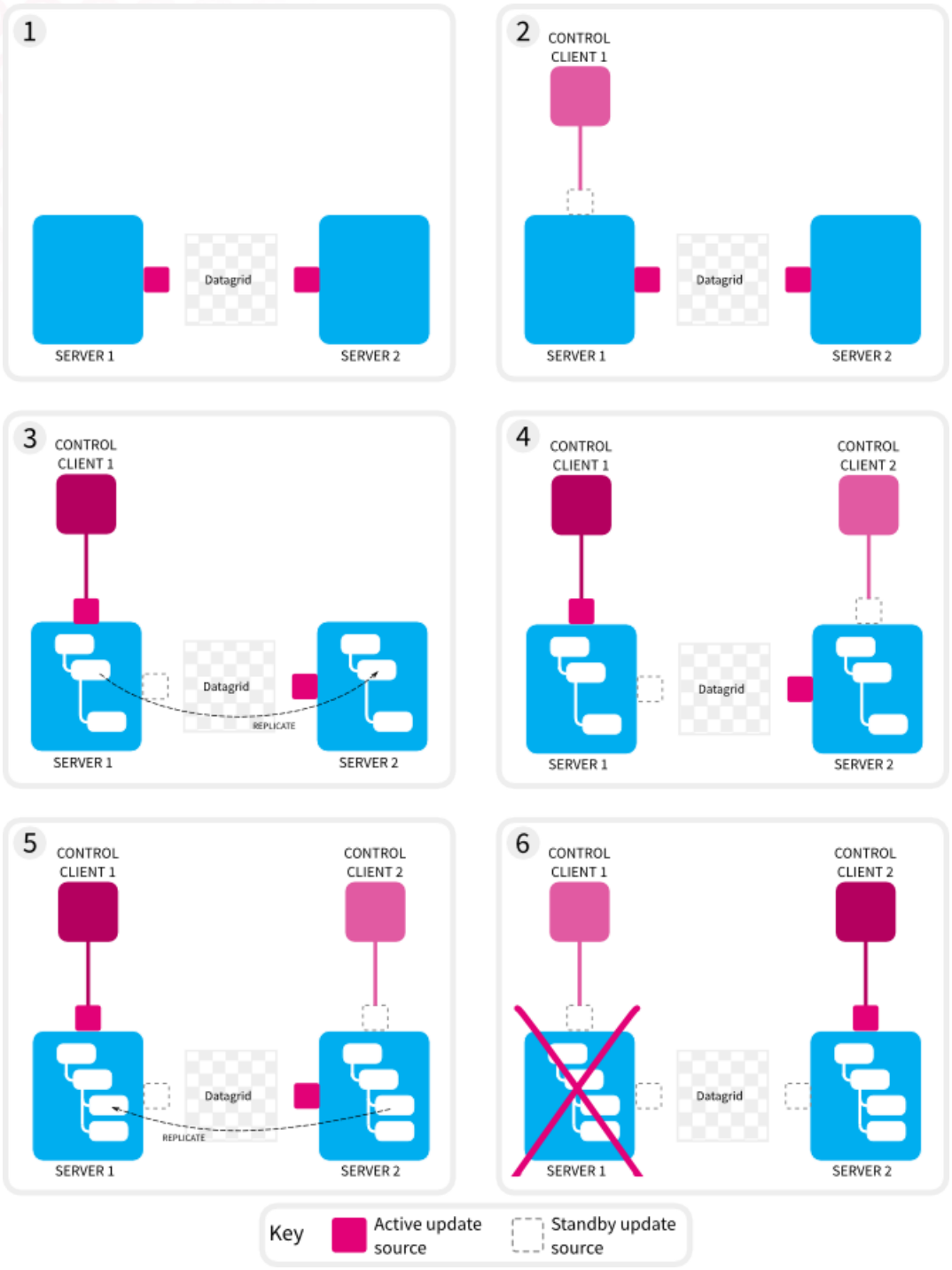


Figure 16: Failover of an active update source

1. Servers that have topic replication enabled for a section of the topic tree create update sources for that section of the topic tree when the server is started.

   These active update sources point into the datagrid and are ready to receive any updates for topics in that section of the topic tree that are reflected into the datagrid.

2. CONTROL CLIENT 1 starts and registers an update source for that section of the topic tree with SERVER 1. This update source is a standby update source. The update source that points at the datagrid remains the active update source.
3. CONTROL CLIENT 1 creates a topic within that section of the topic tree. Through topic replication, the new topic is also created on SERVER 2.

   The update source that SERVER 1 directs at the datagrid becomes a standby update source and the update source that CONTROL CLIENT 1 registered with SERVER 1 becomes active. SERVER 1 sends CONTROL CLIENT 1 a callback to notify it that it is the active update source.

   On SERVER 1, the topics in that section of the topic tree receive their updates from CONTROL CLIENT 1. SERVER 1 reflects this topic data into the datagrid. On SERVER 2, the topics in that section of the topic tree receive their updates from the datagrid.
4. CONTROL CLIENT 2 starts and registers an update source for that section of the topic tree with SERVER 2. The update source for CONTROL CLIENT 2 remains a standby update source and the active update source for SERVER 2 remains pointing at the datagrid.

   The topics on SERVER 2 continue to receive their updates from the datagrid.
5. On SERVER 2, CONTROL CLIENT 2 (or another publisher or control client) can create topics in that section of the topic tree. This does not affect the active update source for that section of the tree.

   The new topic is reflected into the datagrid. Through topic replication, SERVER 1 replicates the new topic in its topic tree. CONTROL CLIENT 1 is the active update source for the new topic.
6. If SERVER 1 becomes unavailable, SERVER 2 becomes the owner of that section of the topic tree and the update source registered by CONTROL CLIENT 2 becomes active. SERVER 2 sends CONTROL CLIENT 2 a callback to notify it that it is the active update source.

   On SERVER 2, the topics in that section of the topic tree receive their updates from CONTROL CLIENT 2. SERVER 2 reflects this topic data into the datagrid.

Considerations

Consider the following factors when using failover of active update sources:

- If the topic paths that the control client uses to register an update source do not match the topic paths configured in the `Replication.xml` configuration file of the server, unexpected behavior can occur.
- Failover of active update sources is not automatic. It must be configured at the server.
- The mechanism that provides failover of active update sources assumes that all servers have the same configuration and that all control clients implement the same behavior as part of a scalable and highly available deployment. If this is not the case, unexpected behavior can occur.
- Do not use topic replication and failover of active update sources on sections of the topic tree that are owned and updated by publishers. Topic updates sent by publishers are not replicated.

Related Links

Updating topics on page 260
A client can use the TopicUpdateControl feature to update topics.

Replication.xml on page 580

This file specifies the schema for the replication properties.

User-written components

Consider which components you must develop to create your solution.

Publishers

Consider whether to develop publishers to distribute data in your solution.

Publishers are written in Java and deployed on the Diffusion server.

You can deploy one or more publishers on a Diffusion server. A publisher can provide the behavior of one or more topics but a topic can belong to only one publisher. The publisher infrastructure is provided by Diffusion and the behavior is provided by the user by writing a publisher.

Why use publishers?

Publishers enable you to manage your topics and updates, and customize their behavior. Unlike clients, publishers are located on the Diffusion server so can communicate more swiftly with the server and do not become disconnected from the server.

Publishers provide the following capabilities:

- Create and delete topics
- Publish updates to topics
- Define topic load data
- Provide topic state to fetch requests
- Send and receive messages to topic paths
- Handle requests for topics that do not exist
- Validate client connections
- Receive notifications of client events
- Subscribe clients to topics

Considerations when using a publisher

Publishers can only be written in Java.

Publisher clients

A publisher can connect upstream to a Diffusion server as if it were a client connecting to that server. The publisher can interact with the server like any other client. When a publisher acts in this way it is said to be acting as a publisher client.

Publisher client capabilities

From within the publisher a publisher client connection to a server is represented by the `PublisherServerConnection` interface. This is a subtype of `ServerConnection` which is the same interface used by connections to servers from clients. A publisher acting as a client has all the same capabilities as any external client. For example, it can subscribe to topics on that server or can send messages to topics on that server.

For full details of the capabilities see the issued API documentation for the ServerConnection interface.

Publisher client connection

A publisher connects to a server listening on a given port at a specified host. The host and port correspond to a client connector configured at that server.

Connecting using API

A publisher can register a connection to a server using `addServerConnection` and then use the `ServerConnection.connect` method to make a connection. Within a publisher the following code creates a new server connection to the localhost on port 8080:

```
PublisherServerConnection conn =
 this.addServerConnection("ServerLocalName","localhost",8080);
```

This connection is named ServerLocalName, this is the name that the publisher uses to refer to the connection. The `addServerConnection` method can take a `ServerDetails` object instead of the host and port.

```
PublisherServerConnection conn =
 this.addServerConnection("ServerLocalName",ConnectionFactory.createServerD
localhost:8080"));
```

See the API documentation for more information about how to use `ConnectionFactory.createServerDetails`.

A failure policy can be set for the connection before connecting which defines the action to take if the connection fails or is lost at some point after connection.

The connection is not opened as soon as it is declared, it must be opened explicitly with `connect`. Once the connection has been opened, the connection can be used to subscribe to topics.

When connected the publisher is notified through the `serverConnected` method and the publisher is able to subscribe to topics on the server as required.

A complete example of how to connect using the API.

```
PublisherServerConnection conn =
 this.addServerConnection("ServerLocalName","localhost",8080);
conn.setFailPolicy(PublisherServerConnectionFailPolicy.RETRY);
try {
 String clientId = conn.connect();
}
catch (APIException e) {
 ...
}
```

Configured connection

It might be more convenient to configure the servers that a publisher automatically connects to in the `etc/Publishers.xml` file. In this case any servers that are declared and connected before `initialLoad` is called, and all connections that have not been explicitly connected to are automatically connected after `initialLoad`.

A failure policy can be configured so that if a connection fails it either stops the publisher or attempt to periodically retry the connection. The default policy is to ignore a failed connection and allow the publisher to deal with it.

Any number of server connections can be configured as elements inside the publisher element of the `etc/Publishers.xml`. The following is a minimal example, all elements of server in it are required. Refer to the XSD for further information.

```
<publisher name="ChildPublisher">
 ...
 <server name="ServerLocalName">
  <host>localhost</host>
  <port>8080</port>
  <input-buffer-size>4k</input-buffer-size>
  <output-buffer-size>64k</output-buffer-size>
  <fail-policy>RETRY</fail-policy>
 </server>
</publisher>
```

Connection failure policy

A connection failure policy can be defined for a server connection to indicate what happens if the connection is lost at some point after the connection is made. For configured connections the failure policy also applies to failure to connect during startup.

A failure policy can be set at any time on a connection using the `setFailPolicy` method.

For configured connections the failure policy can be specified in `etc/Publishers.xml`

The following policy options are available:

DEFAULT

For a configured connection, failure to connect at startup is logged and the publisher can determine the state of such connections after `initialLoad` (for example, in `publisherStarted`) using the `ServerConnection.getState` method.

If a connection is lost, the publisher is notified through the `serverDisconnected` method but no further action is taken.

CLOSE

For a configured connection, failure to connect at startup causes the publisher to be stopped.

For any type of connection, if the publisher loses its connection to the server, the publisher is closed.

RETRY

For a configured connection that fails during startup or any connection that is lost after connection is made, the connection is periodically retried until it is reconnected.

The default retry interval is 5 seconds but this can be specified if required.

Using the connection

Managing a publisher's server connections is done using the name of the connection to identify it.

Any named connection can be retrieved, regardless of whether it was created using the API or the configuration.

```
PublisherServerConnection conn =
 this.getServerConnection("ServerLocalName");
```

Any named connection can be closed and removed from the publisher using `removeServerConnection`.

The publisher can test to see if a name has been used to describe a server connection using `hasServerConnection`.

When a publisher connects to a server the `serverConnected` method is called. This uses the ServerConnection interface to pass the connection information. This method can be overridden so that you can implement custom behavior when a server is connected. Once connected to the server the publisher can interact with its topics, it can subscribe and send messages.

```
@Override
protected void serverConnected(ServerConnection conn) {
 if (conn.getName().equals("ServerLocalName")) {
  conn.subscribe("TopicName");
  ...
 }
}
```

The `serverDisconnected` method is called to notify the publisher a server has disconnected. This uses the ServerConnection interface to pass the connection information. This method can be overridden so that you can implement custom behavior when a server is connected.

```
@Override
protected void serverDisconnected(ServerConnection conn) {
 if (conn.getName().equals("ServerLocalName")) {
  ...
 }
}
```

The publisher is responsible for processing messages that are received from the server using the method. This uses the ServerConnection interface to pass the connection information.

```
@Override
protected void messageFromServer(ServerConnection conn,
 TopicMessage message) {
...
}
```

The ServerConnection can be used to construct and send messages to the server. It provides `createLoadMessage`, `createDeltaMessage` and `send` to communicate with the server. The server treats a publisher client as any other client and no special handling is required on its side.

Notifications

A publisher is notified of a connection to a server using its `serverConnected` method. This is the ideal place to subscribe to topics on the server.

Once connected a publisher receives messages from servers on its `messageFromServer` method which indicates from which server the message came.

A publisher receives notification of the change of status (for example, removal) of any topics it subscribed to on a server using the `serverTopicStatusChanged` method.

The publisher is notified of a lost connection on its `serverDisconnected` method.

Buffer sizes

For a configured connection the sizes of the input and output buffers to use for the socket connection can be specified in `etc/Publishers.xml`

The buffer sizes can also be specified (or changed) for a `ServerConnection` by obtaining the `ServerDetails` (using `getServerDetails`) and using `setInputBufferSize` and setOutputBufferSize. Call these methods before calling connect.

It is important that the buffer sizes specified are compatible with those at the other end of the connection. Remember that the input buffer matches the server's output buffer and the output buffer matches the server's input buffer.

Distribution examples

There are a number of distribution scenarios within Diffusion.

Distributors

This is the term used to represent the situation where there is a single publisher publishing messages to one or more other publishers in other Diffusion Servers.

In this case the one that distributes the messages is known as the distributor publisher and the others are recipients. The recipient publishers connect to the server of the distributor publisher and subscribe to its topics as publisher clients. Messages that the distributor publishes are then published to all clients which in this case are publishers.

This scenario can be used to balance client connections across a number of Diffusion servers all serving the same data.

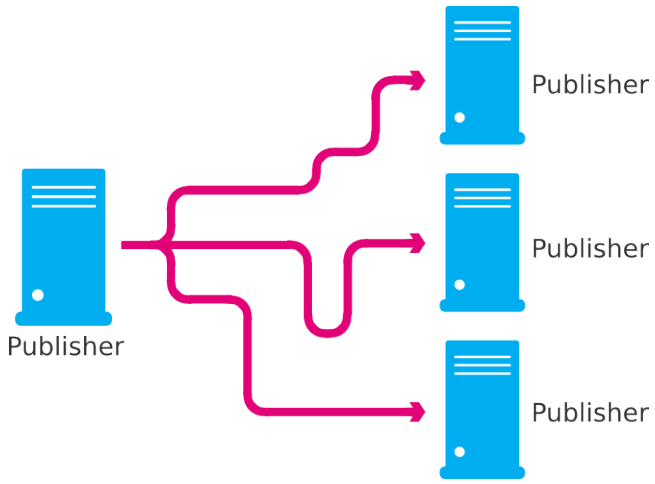


Figure 17: Distributors

Aggregators

Data aggregation

This is the term used to describe the situation where a single publisher aggregates messages received from one or more other publishers.

In this case the aggregator connects to more than one other Diffusion server and subscribes to topics on them.

This scenario can be used to aggregate data being received from more than one source providing a single point of contact for clients which see only the aggregated data.

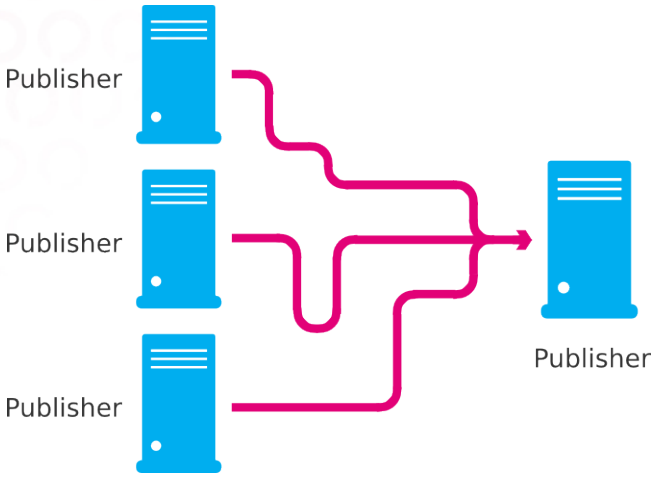


Figure 18: Aggregators

Mixed mode

The client/server relationship between publishers is not restricted to the above examples. You can set up a peer-to-peer network of publishers, which communicate with each other in any way required.

A publisher publishes to clients. Clients connected to a Diffusion server can be a mixture of publisher clients and other clients.

Clients

Consider how you use clients in your solution.

Clients are key to a Diffusion solution. Your solution must include clients as an endpoint to distribute data to. However, clients can also be used for control purposes.

When using clients in your solution, consider the following:

- What types of client you require
- What you use your clients for

Client types

Diffusion provides APIs for many languages and platforms. Some of these APIs have different levels of capability. You can write a client that uses one of these APIs to connect to the Diffusion server or you can write a client that uses the Diffusion protocol for TCP (DPT) to communicate directly with the Diffusion server.

A client's type is a combination of the API it uses and the protocol it uses to connect to the Diffusion server.

APIs

JavaScript Unified API

Use this API to develop browser or Node.js clients that can have control capabilities.

Java Unified API

Use this API to develop Java clients that can have control capabilities.

.NET Unified API

Use this API to develop clients in C# that can have control capabilities.

C Unified API

Use this API to develop C clients that can have control capabilities.

iOS Classic API

Use this API to develop mobile clients in Objective-C that do not have control capabilities.

Android Classic API

Use this API to develop mobile clients in Java that do not have control capabilities.

Silverlight Classic API

Use this API to develop browser clients that do not have control capabilities.

Flash Classic API

Use this API to develop browser clients in ActionScript that do not have control capabilities.

JavaScript Classic API

You can use this API to develop browser clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

Java Classic API

You can use this API to develop Java clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

.NET Classic API

You can use this API to develop C# clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

C Classic API

You can use this API to develop C clients that do not have control capabilities. However, we recommend that you use the Unified API instead.

Publisher clients

Publisher clients are publishers deployed on a Diffusion server that connect to another Diffusion server as a client. You can use the Java Publisher API to develop a publisher.

Protocols

The following protocols, and their secure versions, are available:

DPT

The DPT protocol (Diffusion protocol over TCP) creates a TCP connection and uses it to send and receive messages in a full duplex way. Load balancers treat these connections as TCP connections.

WebSockets

The WebSocket implementation provides a browser-based full duplex connection, built on top of WebSocket framing. This complies with the WebSockets standards and is usable with any load balancer or proxy with support for WebSockets.

HTTP Full Duplex

HTTP Full Duplex acts like HTTP in the initial connection handshake and acts like DPT for the exchange of messages. HTTP Full Duplex wraps the Diffusion protocol with HTTP request and response headers. Unlike true HTTP, it operates in a full duplex manner. For example, the client can send a response that does not correspond to a request.

This is acceptable to a number of network intermediaries (load balancers and firewalls), and can be a pragmatic way to communicate over a single bi-directional, end-to-end connection via intermediaries that do not accept the WebSocket.

However, if the API supports WebSockets, we recommend that you use WebSockets instead.

HTTP Chunked Streaming

HTTP Chunked Streaming provides a streaming connection for messages from the server by using HTTP chunked encoding. A separate TCP connection is used to send messages from the client to the server. This provides two simplex connections, one based on request/response (upstream) and the other streaming data (downstream). This relies on HTTP/1.1 so ensure that network intermediaries such as load balancers are HTTP/1.1 aware.

HTTP Polling

HTTP polling uses HTTP to make a long poll request. Each request remains open until a message is available. More than one message will be returned if available. A separate TCP connection is used to send messages from the client to the server.

Table 14: Supported protocols by client

Client	DPT	DPTS	WS	WSS	HTTP Full Duplex	HTTPS Full Duplex	HTTP Chunked Streamin	HTTPS Chunked Streamin	HTTP Polling	HTTPS Polling
JavaScript Unified API	✓ [3]	✓ [3]	✓	✓						
Java	✓	✓	✓	✓	✓	✓				
.NET	✓	✓	✓	✓	✓	✓				
JavaScript Classic API	✓ [3]	✓ [3]	✓	✓			✓ [4]	✓ [4]	✓	✓
Flash	✓	✓					✓	✓	✓	✓
Silverlight	✓	✓							✓	✓
iOS	✓	✓	✓	✓						
Android	✓	✓								
C	✓	✓								
Publisher client	✓ (Recommended)	✓	✓	✓	✓	✓				

[1] [2]

Using clients

Most clients connect to the Diffusion server only to subscribe to topics and receive message data on those topics. Some clients can also perform control actions such as creating and updating topics or handling events.

Subscribe to topics and receiving data

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, C Unified API, JavaScript Classic API, Java Classic API, .NET Classic API, C Classic API, iOS Classic API, Android Classic API, Flash Classic API, Silverlight Classic API

The majority of clients that connect to the Diffusion server, do so to subscribe to topics and receive updates that are published to those topics. These are the clients used by your customers to interact with the data your organization provides.

Control Diffusion, other clients, or the data

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, C Unified API

You can also develop clients that control aspects of the Diffusion server, other clients, or the data distributed by Diffusion. These are the clients that are used by users or systems inside your organisation.

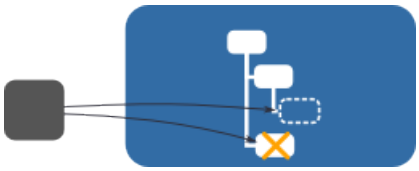
Using clients for control

Clients can perform control actions that affect the Diffusion server, other clients, or the data distributed by Diffusion.

Supported in: JavaScript Unified API, Java Unified API, .NET Unified API, C Unified API

When designing your Diffusion solution, decide whether you want to use clients to perform the following actions:

Create and delete topics



Clients can create any type of topic on the Diffusion server. These topics can be created explicitly or dynamically in response to a subscription request from another client.

These topics have the lifespan of the Diffusion server unless the client specifies that the topic be removed when the client session closes.

Clients can also delete topics from the Diffusion server.

You can also use publishers to create and delete topics.

For more information, see Managing topics on page 220.

[1] Supported by Flash/Silverlight
[2] Supported natively and by Flash

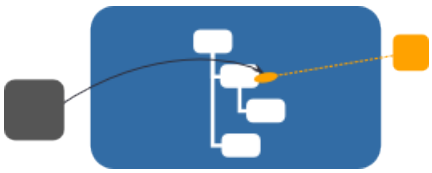
Publish updates to topics

Clients can publish updates to topics that are pushed out to clients subscribed to that topic. These updates can be made exclusively, so that only one client can update a given topic, or non-exclusively, allowing any client to update a given topic.

You can also use publishers to publish updates to topics.

For more information, see Updating topics on page 260.

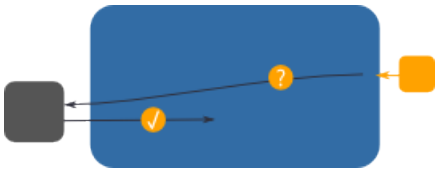
Subscribe other clients to topics



Clients can subscribe other client sessions to topics and also unsubscribe other client session from topics.

For more information, see Managing subscriptions on page 344.

Authenticate other clients



Clients can provide authentication decisions about whether to allow or deny other client sessions connecting to the Diffusion server. These clients can also assign roles to the connecting client sessions that define the permissions the connecting client has.

You can also use the system authentication handler or an authentication handler located on the Diffusion server to authenticate other clients.

For more information, see User-written authentication handlers on page 145.

Modify the security information stored on the Diffusion server

Clients can modify the information stored in the security store on the Diffusion server. The security store can be used to specify which permissions are assigned to roles and which roles are assigned to anonymous sessions, and named-principal sessions.

You can also use publishers to modify security information stored on the Diffusion server.

For more information, see Updating the security store on page 326.

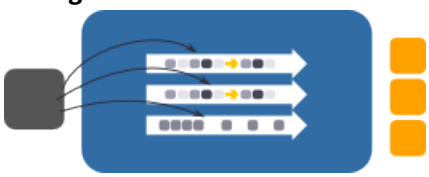
Modify the authentication information stored on the Diffusion server

Clients can modify the information stored in the system authentication store on the Diffusion server. The system authentication store can be used to specify which principals a client session can use to connect and what roles are assigned to an authenticated client session.

You can also use publishers to modify authentication information stored on the Diffusion server.

For more information, see Updating the system authentication store on page 313.

Manage the flow of data to clients



Updates are pushed to subscribing clients through client queues. Clients can receive notifications when client queues reach a certain threshold. These clients can manage the client queues by turning on throttling or conflation for the queue.

For more information, see Managing clients on page 339.

To handle messages sent to topic paths by clients and send messages to specific clients

Clients can send messages through topic paths to specific clients. Clients can also register to handle messages that are sent to a topic path. Messages sent using topic paths do not update the topic.

You can also use publishers to handle messages on topic paths and send messages to clients.

For more information, see Messaging to clients on page 289.

Other user-written components

Diffusion provides many opportunities to create user-written components that define custom behavior. Consider whether to develop any of these components as part of your solution.

Server-related components

All of these components must be created as Java classes and put on the classpath of the Diffusion server.

Authentication handlers

These components handle authentication of clients that connect to the Diffusion server or change the principal that they use to connect to the Diffusion server. If the client connection is allowed, the authentication handler assigns roles to the client session.

You can have zero, one, or many authentication handlers configured on your Diffusion server.

For more information, see Developing a local authentication handler and Developing a composite authentication handler.

> **Note:** Local authentication handlers, on the Diffusion server, can be written only in Java. However, control authentication handlers that are part of a client whose API supports Authentication Control can be written in other languages.

Hooks

Startup and shutdown hooks are called by the Diffusion server. The startup hook is instantiated and called as the Diffusion server starts and before publishers are loaded. The shutdown hook is called as the Diffusion server stops.

For example, you can use a shutdown hook to persist some aspect of the state of the Diffusion server to disk.

HTTP service handlers

These components handle HTTP requests as part of an HTTP service in the Diffusion server's built-in web server. Provide a user-written HTTP service handler to enable the Diffusion web server to handle any kind of HTTP request.

Thread pool handlers

These handlers define custom behavior in the Diffusion server related to the inbound thread pool.

You can provide a rejection handler that customizes the behavior when a task cannot be run by the thread pool. By default, if a task cannot be run by the inbound thread pool - for example, if the thread pool is overloaded - the rejected task is run in the calling thread.

You can provide a notification handler that receives notifications when events occur on the inbound thread pool.

DEPRECATED: Authorization handlers

Authorization handlers are deprecated and have been replaced by role-based authorization. For more information, see Role-based authorization on page 134.

An authorization handler controls authorization and permissioning for clients and users. You can have zero or one authorization handler configured on your Diffusion server.

Authorization handlers can be written only in Java.

Topic- and data-related components

All of these components must be created as Java classes and put on the classpath of the Diffusion server.

Message matchers

Message matchers are used to customize conflation behavior. These classes that define how the Diffusion server locates messages on a client's message queue that are to be conflated.

By default, messages for conflation are matched if they are on the same topic.

For more information, see Conflation on page 76.

Message mergers

Message mergers are used to customize conflation behavior. These classes that define how the Diffusion server conflates matching messages.

By default, the older of the matching messages is removed.

For more information, see Conflation on page 76.

Comparators and collators

These components define the behavior of an ordered paged topic. Comparators implement `java.util.Comparator` and can customize the ordering of lines in a paged string or paged record topic. Collators implement `java.text.RulesBased Collator` and can define the ordering of lines in a paged record topic.

For more information, see Creating paged topics on page 221.

Custom field handlers

These components handle the data in custom fields of your record topics. A custom field handler can define the default value of a custom field, parse incoming data into the format required by the custom field, and compare data in custom fields of the same type for equality.

For more information, see Metadata on page 61.

Custom topic data handlers

These components handle the behavior of a custom topic. When you create a custom topic, you provide a custom topic handler that defines how the topic data is maintained, compared, and sent to subscribing clients.

For more information, see Custom topics on page 56.

Routing topic handlers

These components handle the behavior of a routing topic. When you create a routing topic, you provide a routing topic handler that, when a subscription to the routing topic is made, maps the routing topic to another topic on the Diffusion server on a client-by-client basis.

For more information, see Routing topics on page 55.

Service handlers

These components handle the behavior of a service topic. When you create a service topic, you provide a service handler that receives requests from clients on the topic and provides responses through the topic.

For more information, see Service topics on page 59.

Third party components

Diffusion interacts with a number of third-party components. Consider how you use these components as part of your solution.

Web servers

Consider how to use web servers as part of your Diffusion solution.

While the Diffusion server includes a web server, this internal web server is intended to be used to host the Diffusion console or to host a static page you can use for uptime testing.

Do not use the Diffusion web server as the host for your production website. Instead use a third-party web server.

Why use a web server?

You can use a third-party web server to host your Diffusion browser clients.

Considerations when using a web server

If your web server hosts a client that makes requests to a Diffusion server in a different URL space, use a load balancer to composite the URL spaces and interoperate with browser security.

Load balancers

Consider how you use load balancers in your solution.

We recommend that you use load balancers in your Diffusion solution.

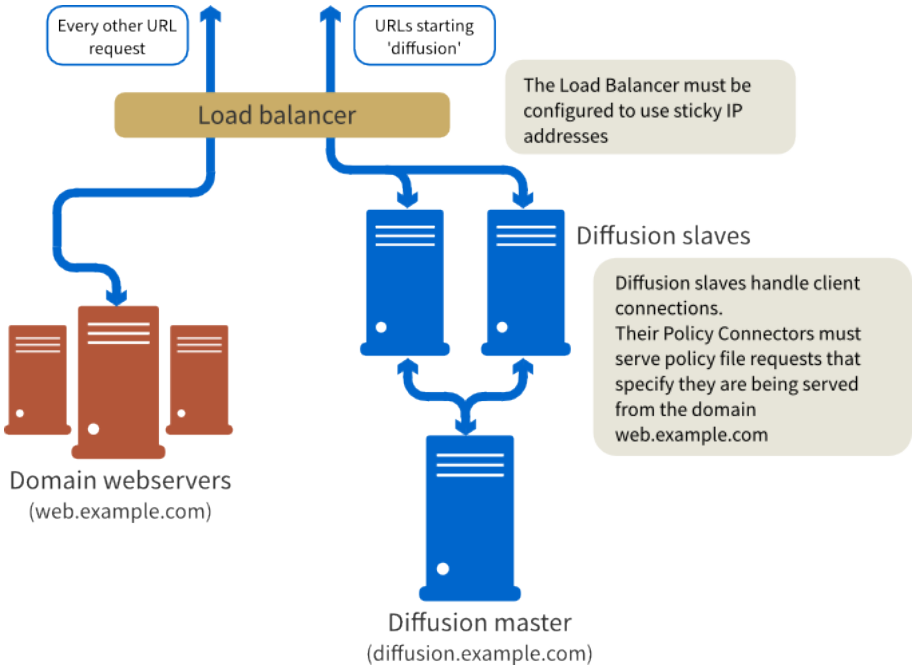
Why use load balancers?

Balancing client traffic across multiple Diffusion servers

Distribute incoming requests from clients fairly over the Diffusion servers in your solution and ensure that all traffic for a specific client is routed to the same Diffusion server.

Compositing URL spaces

If your Diffusion servers are located at a different URL to the Diffusion browser clients hosted by your web servers, you can use a load balancer to composite the URL spaces. This enables Diffusion solutions to interoperate with browser security.


Every other URL request
URLs starting 'diffusion'
Load balancer
The Load Balancer must be configured to use sticky IP addresses
Diffusion slaves
Diffusion slaves handle client connections.
Their Policy Connectors must serve policy file requests that specify they are being served from the domain web.example.com
Domain webservers
(web.example.com)
Diffusion master
(diffusion.example.com)

SSL offloading

Diffusion clients can connect to your solution using TLS, commonly called secure socket layer (SSL). The TLS can terminate at your load balancer or at your Diffusion server. Terminating the TLS at the load balancer reduces CPU cost on your Diffusion servers.

Considerations when using load balancers

Do not use connection pooling for connections between the load balancer and the Diffusion server. If multiple client connections are multiplexed through a single server-side connection, this can cause client connections to be prematurely closed.

In Diffusion, a client is associated with a single TCP/HTTP connection for the lifetime of that connection. If a Diffusion server closes a client, the connection is also closed. Diffusion makes no distinction between a single client connection and a multiplexed connection, so when a client sharing a multiplexed connection closes, the connection between the load balancer and Diffusion is closed, and subsequently all of the client-side connections multiplexed through that server-side connection are closed.

Multiple users masquerading behind a proxy or access point can have the same IP address, and all requests from clients with that IP address are routed to the same Diffusion instance. Load balancing still occurs, but some hosts might be unfairly loaded.

Proxies

Consider how to use proxies as part of your solution.

Clients can connect to the Diffusion server through an HTTP proxy by using the HTTP `CONNECT` verb to create the connection and tunneling any of the supported transports through that connection.

Considerations when using a proxy

Clients that use the Classic API to connect to the Diffusion server, can connect through a proxy only if the proxy does not require authentication.

JMS

Consider whether to incorporate JMS providers into your solution.

If a third-party JMS provider is part of your solution, you can map JMS queues and topics to Diffusion topics by using the Diffusion JMS adapter.

We support integration with JMS providers that conform to version 1.1 or later of the JMS specification.

The following JMS products have been tested and are certified by Push Technology for use with the JMS adapter:

- Apache ActiveMQ v5.11
- IBM MQ v8

Why use a third-party JMS provider

If you are already using a JMS provider to move data in your internal system, you can integrate it with Diffusion to distribute that data to clients and users outside of your organization.

Diffusion JMS adapter

The JMS adapter for Diffusion, enables Diffusion clients to transparently send data to and receive data from destinations (topics and queues) on a JMS server. It is highly configurable and can support the following scenarios:

Pubsub

Messages on a JMS destination can be published to a Diffusion topic. For more information, see Publishing using the JMS adapter on page 109.

Messaging

Messages can be sent between JMS destinations and Diffusion clients.

- A message on a JMS destination can be relayed to a Diffusion client through a topic path.
- A Diffusion client can send a message on a topic path that is sent to a JMS destination.

For more information, see Sending messages using the JMS adapter on page 110.

Request-response

The JMS provider can integrate with services that interact using an asynchronous request-response pattern. Diffusion exposes these JMS services through its messaging capabilities. For more information, see Using JMS request-response services with the JMS adapter on page 112.

Data that flows between JMS and Diffusion must be transformed. JMS messages contain headers, properties, and a payload. Diffusion messages contain just content. For more information about how data is transformed between JMS and Diffusion, see Transforming JMS messages into Diffusion messages or updates on page 106.

Considerations when using the JMS adapter

Note: If you currently use the legacy JMS adapter version 5.1, you must reimplement to use this JMS adapter. No migration path for configuration is available.

You cannot currently publish data to a Diffusion topic and have it sent to a JMS destination.

Only textual content and JMS TextMessages are supported. Binary content is not currently supported.

Data must be transformed between JMS messages and Diffusion content.

If multiple Diffusion servers subscribe to the same JMS queue in a request-response scenario, there is the risk of one server consuming messages intended for another server. Use JMS selectors to ensure that the JMS adapter only receives those messages intended for it.

The creation of temporary queues and topics by the JMS adapter is not currently supported.

Dynamic changes to the JMS adapter configuration are not supported.

Durable subscriptions are not supported.

JMS transactions are not supported.

The only acknowledgment mode that is supported is AUTO_ACKNOWLEDGE.

Session properties are not currently supported. The exception is the `$Principal` property.

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Transforming JMS messages into Diffusion messages or updates

JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

The following modes of transformation are provided:

Basic

Only the textual content of a message is relayed.

JSON

All JMS headers and properties are relayed, in addition to the textual content of the message. These values are expressed as JSON in the corresponding Diffusion message.

You can configure which one of these transformation modes your JMS adapter uses at the per topic level.

JMS message structure

JMS messages comprise headers, properties, and a payload. Currently, only JMS TextMessages are supported by the JMS adapter.

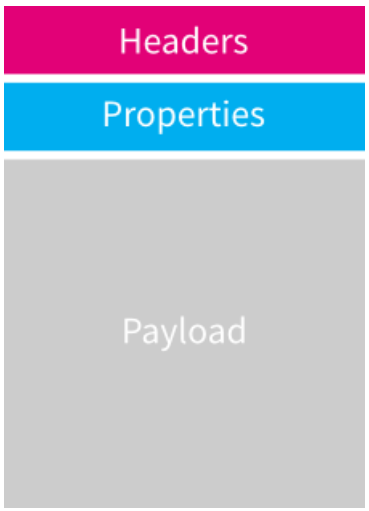


Figure 19: JMS message structure

Headers

This is a fixed set of properties whose names all begin with 'JMS'. Some, such as JMSDestination, are mandatory. Others are optional. For more information, see https://docs.oracle.com/javaee/7/api/javax/jms/Message.html.

Properties

A set of name-value pairs.

Payload

The contents of the message. This is a String.

Basic transformation

In a basic transformation only the textual payload or content of the message is relayed in either direction.

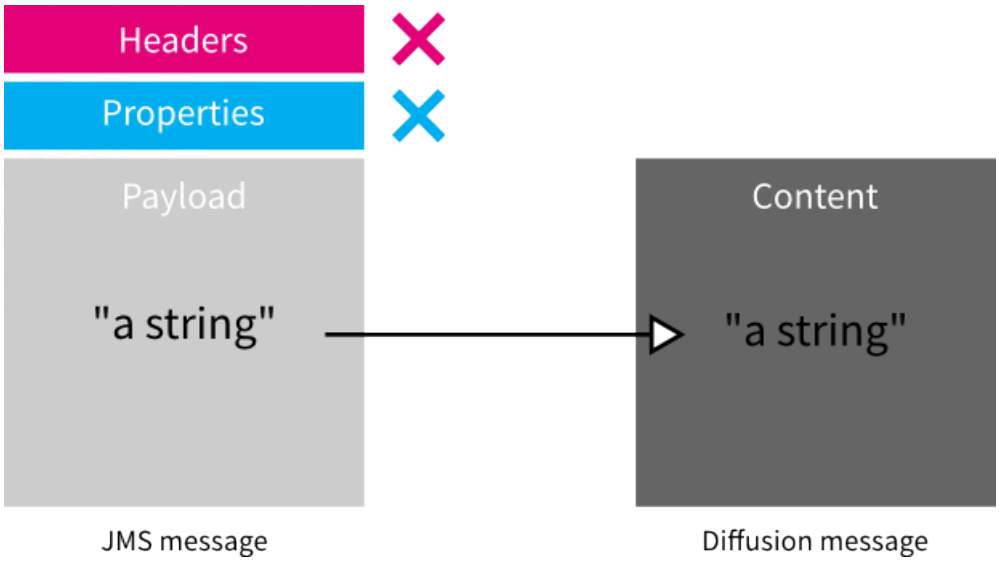


Figure 20: Basic mapping from a JMS message to a Diffusion message

When relaying a JMS message to Diffusion, the JMS adapter creates a Diffusion message whose content is the JMS message payload. The headers and properties of the JMS message are ignored.

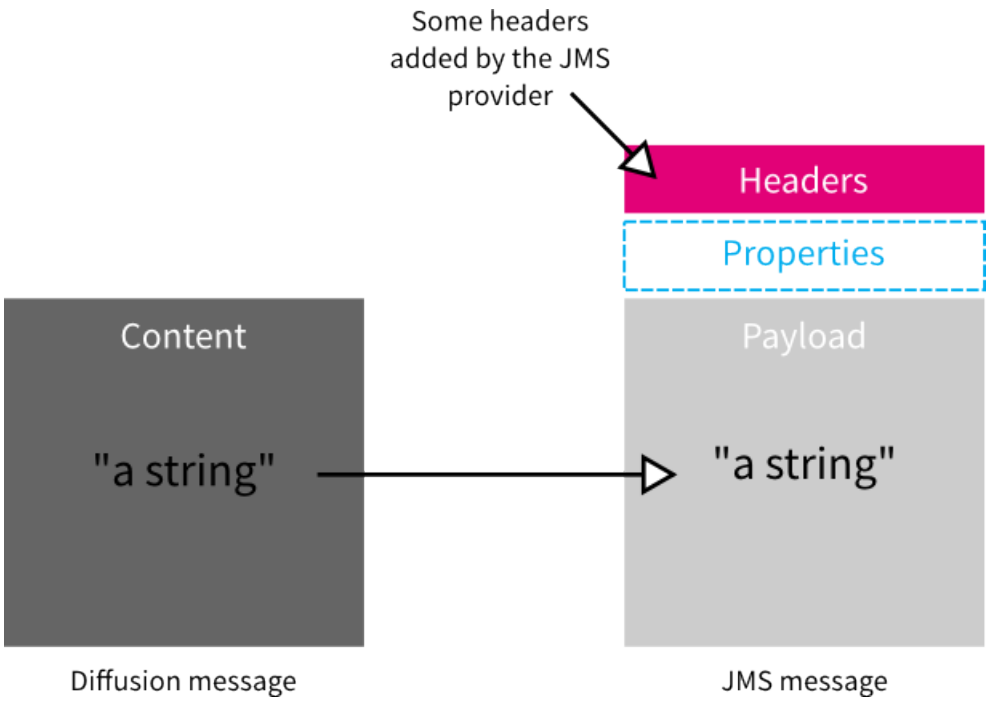


Figure 21: Basic mapping from a Diffusion message to a JMS message

When relaying a Diffusion message to JMS, the JMS adapter sets the JMS message payload to be the Diffusion content. The JMS adapter does not set any properties or headers on the JMS message. The JMS provider sets any mandatory headers that are required on the JMS message.

JSON transformation

In a JSON transformation all information is relayed both directions. The JMS message information is expressed in JSON format inside the Diffusion message content.

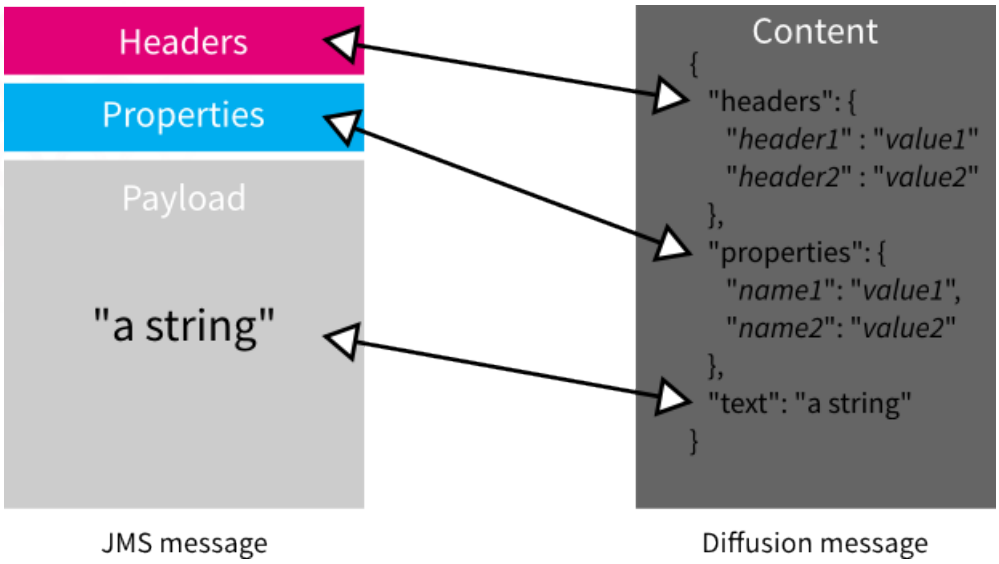


Figure 22: Mapping from a JMS message to and from JSON in a Diffusion message

When relaying a JMS message to Diffusion, the JMS adapter performs the following actions:

- Expresses the Diffusion content as a single JSON object.
- Maps the JMS message headers to a JSON object called “headers” inside of the Diffusion message content. The “headers” object contains all of the JMS message headers as name-value pairs. For example,

```
"headers": {
  "JMSType": "abc",
  "JMSPriority": 9
}
```

- Maps the JMS message properties to a JSON object called “properties” inside of the Diffusion message content. The “properties” object contains all of the JMS message properties as name-value pairs. For example,

```
"properties": {
  "AString": "def",
  "ABoolean": true
}
```

- Maps the textual payload of the JMS message to a JSON item called “text” inside of the Diffusion message content. For example,

```
"text": "Message content"
```

When relaying a Diffusion message to JMS, the JMS adapter parses the JSON content of the Diffusion message and uses the information to set the headers, properties, and payload of the JMS message accordingly.

Related Links

Configuring the JMS adapter on page 583

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Publishing using the JMS adapter

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

Publishing data from a JMS destination onto a Diffusion topic

You can configure the JMS adapter to subscribe to a JMS destination and to associate that subscription with a Diffusion topic.

The Diffusion topic can be stateful or stateless, but stateful topics must be created with an initial value. For more information, see Example: Configuring topics for use with the JMS adapter on page 593.

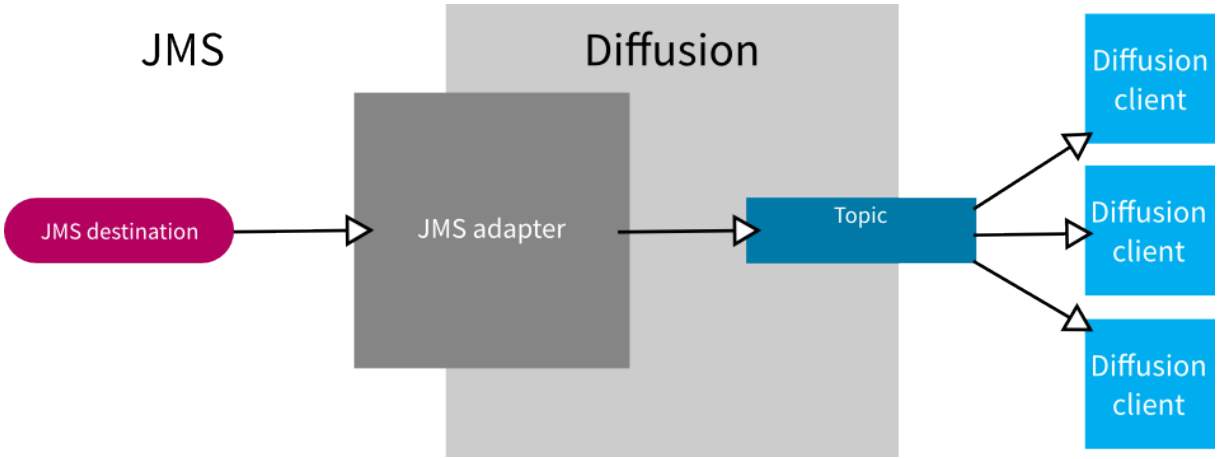


Figure 23: JMS adapter: Publishing from JMS to Diffusion

1. A message is published to the JMS destination.
2. The JMS adapter receives the JMS message.
3. The JMS adapter transforms the JMS message into a Diffusion message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
4. The JMS adapter publishes the transformed message to the Diffusion topic.
5. Diffusion clients that are subscribed to the Diffusion topic receive the transformed message.

Publishing data from a Diffusion topic to a JMS destination

This is not currently supported.

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring pubsub with the JMS adapter on page 594
Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

Sending messages using the JMS adapter

The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

Sending a message from a Diffusion client to a JMS destination

You can configure the JMS adapter to handle messages sent on a Diffusion topic path and to associated messages received on that topic path with a JMS destination.

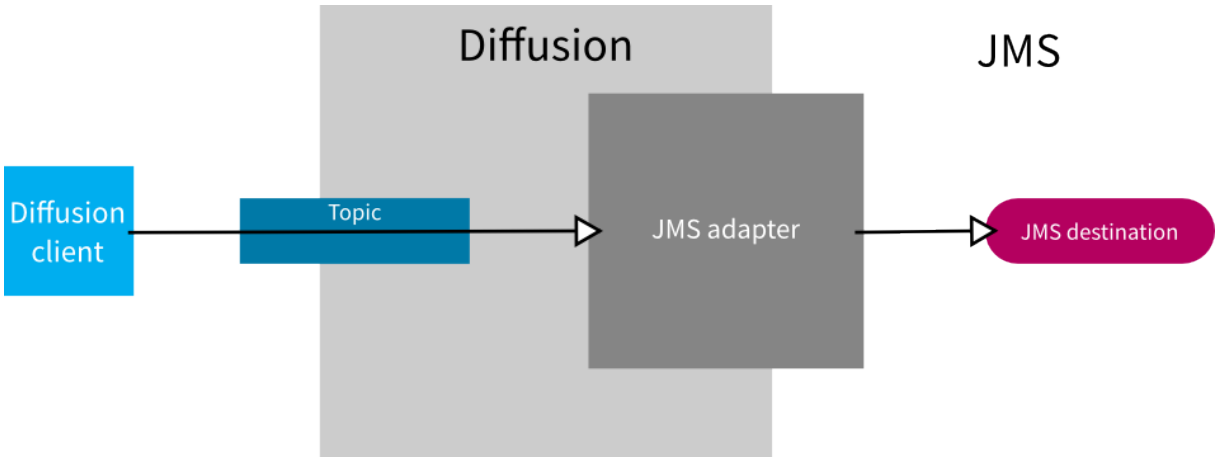


Figure 24: JMS adapter: Message flow from Diffusion to JMS

1. A Diffusion client sends a message to a topic path.
2. The JMS adapter receives the message.
3. The JMS adapter transforms the Diffusion message into a JMS message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
4. The JMS adapter sets a JMS header or property to include the Diffusion server name of the JMS adapter and the session ID of the Diffusion client.

   This header or property is used as a return address for any response messages and is nominated using the `routingProperty` configuration element. By convention, `JMSCorrelationID` is often used. For more information, see JMSAdapter.xml.
5. The JMS adapter publishes the transformed message to the JMS destination.

Sending a message from a JMS destination to a Diffusion client

You can configure the JMS adapter to subscribe to a JMS destination and to associate that subscription with a Diffusion topic path to send a message through.

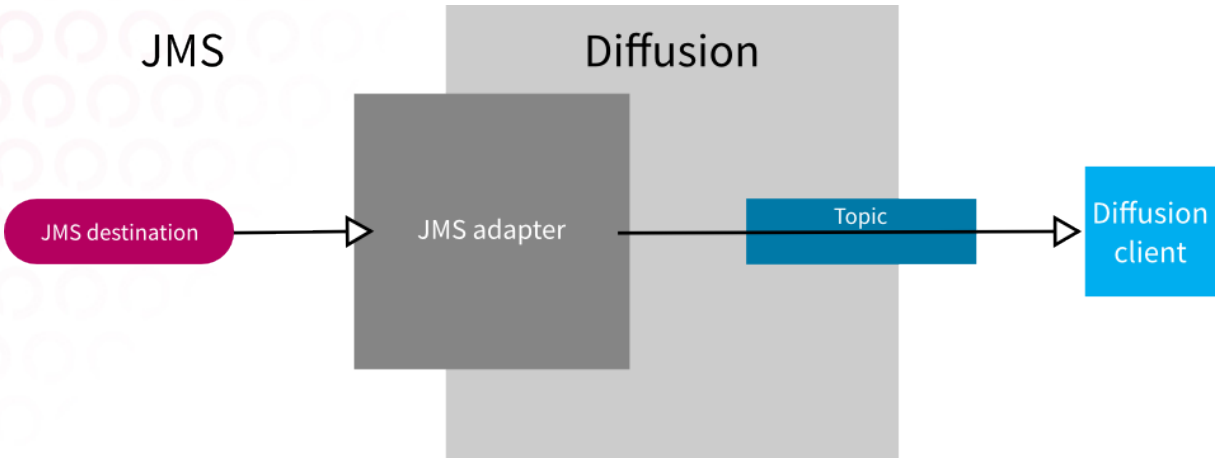


Figure 25: JMS adapter: Message flow from JMS to Diffusion

1. The JMS adapter receives a message from a JMS destination.
2. The JMS adapter transforms the JMS message into a Diffusion message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
3. The JMS adapter checks the nominated JMS header or property for the server name and session ID of the recipient client.

   This header or property is nominated using the `routingProperty` configuration element. For more information, see JMSAdapter.xml.
4. The JMS adapter sends the transformed message through the topic path to the recipient client session.

Error scenarios

- The JMS adapter consumes a message from a JMS destination that is not intended for it. That is, the routing property or header does not contain the Diffusion server name of the JMS adapter.

  In this case, the JMS adapter drops the message and logs the failure to deliver.

  You can avoid this scenario by using a JMS selector when subscribing to the JMS destination that specifies the JMS adapter is only interested in messages whose routing property or header include its Diffusion server name.
- The JMS adapter receives a message from a Diffusion client, but cannot send it on to JMS because the JMS provider is not connected.

  In this case, the JMS adapter returns the message to the client on the same topic and logs the failure to deliver.
- The JMS adapter receives a message from a JMS destination, but cannot send it on to the Diffusion client because the Diffusion client is not connected.

  In this case, the JMS adapter drops the message and logs the failure to deliver.

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring messaging with the JMS adapter on page 595
Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

Using JMS request-response services with the JMS adapter

You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

Exposing a JMS service through Diffusion messaging is a typical use case for the JMS adapter.

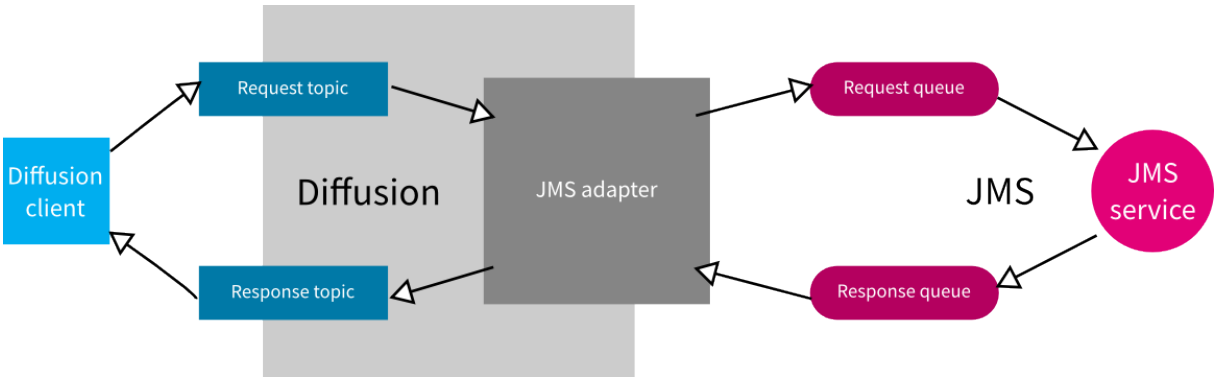


Figure 26: JMS adapter: Request-response message flow

1. A Diffusion client sends a message to a Diffusion topic path configured in the JMS adapter to receive service requests.
2. The JMS adapter receives the message on the request topic path.
3. The JMS adapter transforms the Diffusion message into a JMS message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
4. The JMS adapter adds a routing property or header to the JMS message identifying the Diffusion server and client to return a response to. This return information is of the form *diffusion_server_name*/*diffusion_client_session_id*.
5. The JMS adapter sends the message to the JMS service request queue.
6. The JMS service receives the request.
7. The JMS service acts on the request.
8. The JMS service places a response message on its response queue. This message must include the routing property or header that identifies the Diffusion server and client to return the response to.
9. The JMS adapter receives the response message from the JMS response queue.
10. The JMS adapter transforms the response message into a Diffusion message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
11. The JMS adapter uses the information in the routing property or header to discover the connected client session to relay the response to.
12. The JMS adapter sends the response message to the Diffusion client through a topic path.

Error scenarios

- The JMS adapter consumes a message from a JMS service response queue that is not intended for it. That is, the routing property or header does not contain the Diffusion server name of the JMS adapter.

  In this case, the JMS adapter drops the message and logs the failure to deliver.

  You can avoid this scenario by using a JMS selector when subscribing to the JMS queue that specifies the JMS adapter is only interested in messages whose routing property or header include its Diffusion server name.
- The JMS adapter receives a message from a Diffusion client, but cannot send it on to JMS because the JMS provider is not connected.

  In this case, the JMS adapter returns the message to the client on the same topic and logs the failure to deliver.
- The JMS adapter receives a message from a JMS destination, but cannot send it on to the Diffusion client because the Diffusion client is not connected.

  In this case, the JMS adapter drops the message and logs the failure to deliver.

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring the JMS adapter to work with JMS services on page 596
Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

DEPRECATED: Legacy JMS adapter

For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

The version 5.1 JMS adapter is packaged in the file `JMSAdapter51.jar` and configured using the `JMSAdapter51.xml` configuration file.

Note: We recommend that you do not develop new solutions using the version 5.1 JMS adapter. Instead use the latest version.

The JMS Adapter for Diffusion, enables Diffusion clients to transparently send and receive messages with topics and queues on a JMS server.

The behavior of a JMS adapter is configured in XML. No coding is required to use a JMS adapter.

The Diffusion server maintains a topic tree. Configure the JMS adapter to map a branch of the topic tree beneath a root topic to JMS topics and queues on your third-party JMS provider.

Considerations when using the JMS adapter

Dynamic topics

Diffusion topics are created dynamically and do not exist until a valid JMS subscription has been made and data received. Mapping to JMS queues is performed in the same way.

Message delivery

Delivery of messages to clients subscribing to a Diffusion topic that is mapped to a JMS queue is different to standard message delivery. Instead it is in keeping with the delivery characteristics of JMS queues.

When there are multiple Diffusion clients listening for data originating from the same JMS queue, each message is delivered to at most one client. Depending on the configuration of the JMS adapter the receiving client is chosen either randomly or based on the client with the fewest number of messages waiting for delivery.

Wildcards

You cannot subscribe to JMS topics using wildcards or topic selectors.

Temporary topics and queues

A common use for temporary queues is to set up a return path for request-reply operations.

A Diffusion client can request access to a JMS temporary topic or queue in the same way as subscribing to a JMS destination.

In the topic tree, temporary topics exist as sub-topics under the jms/tmp/topic topic.

Acknowledgment

The only acknowledgment mode supported is AUTO_ACKNOWLEDGE. This implies that when any message is received from the JMS server, an acknowledgment is sent from the JMS adapter to the JMS server.

Acknowledgments sent from a Diffusion client to the JMS server (CLIENT_ACKNOWLEDGE) are not supported.

Message headers

The JMS adapter copies the standard JMS headers and user-supplied headers when converting between JMS and Diffusion message types.

In Diffusion, headers are logically grouped in pairs. Property *n* is the name, and *n+1* is the value, where *n* is an even number. In the case of the `JMSReplyTo header`, the related header `DiffusionReplyTo` is created. Mapping between a `JMSReplyTo` destination and a `DiffusionReplyTo` topic is handled transparently by the adapter.

In most circumstances, a Diffusion client can ignore the `JMSReplyTo` header. The header is forwarded to the client for completeness.

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

JMS adapter data flow examples

The examples in this section show how data flows between the Diffusion server and a JMS provider.

The following scenarios assume that the JMS adapter is configured with a root topic of jms.

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

Receiving data from JMS

Diffusion clients can receive data from a JMS provider through the JMS adapter. A client can receive updates from a JMS topic or messages from a JMS queue.

Receiving updates from a JMS topic

This section shows how a client receives updates from the JMS topic XYZ.

1. Diffusion client creates a subscription to the topic `jms/topic/XYZ`.
2. Once a message has been sent from the source system into the JMS server, the Diffusion client receives an initial topic load message.
3. Subsequent messages from the source system result in delta messages being delivered to the Diffusion client.

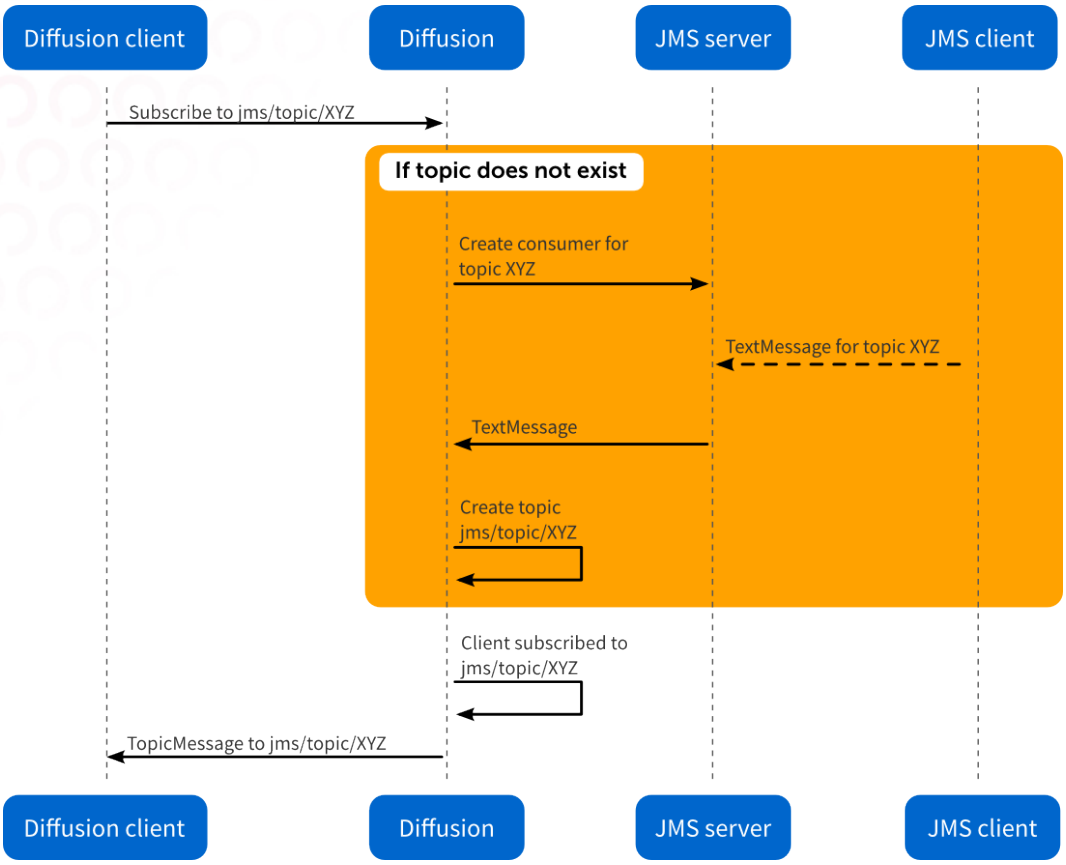


Figure 27: Subscription flow

Receiving messages from a JMS queue

The same process occurs for receiving messages from JMS queues, with the following differences:

- Other clients subscribing to the same JMS queue (either through Diffusion or directly using JMS) might receive the message instead of our client.
- All messages originating from JMS queues are initial topic load messages. Since a sequence of messages from a JMS queue are unlikely to always be delivered to the same client, the concept of delta messages does not readily apply and the full message state must be supplied every time.

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

Sending messages to JMS
Diffusion clients can send messages to a JMS provider through the JMS adapter.

1. The Diffusion client sends a message to the topic path jms/topic/XYZ.

2. The JMS server receives an equivalent `TextMessage` on the XYZ topic.

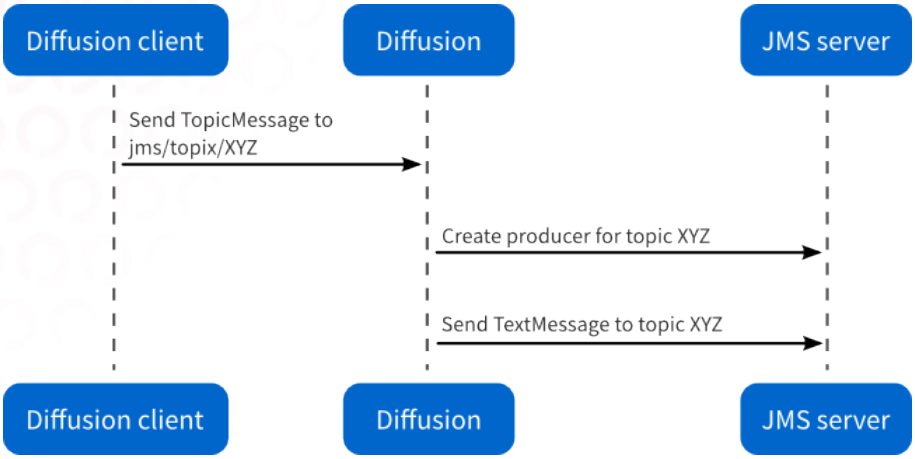


Figure 28: Sending flow from a Diffusion client to a JMS topic (or queue)

Unlike some Diffusion solutions, it is not necessary to subscribe to a Diffusion topic before sending it a message which targets a JMS destination.

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

Processing a request-reply message with a Diffusion client

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

Typically, the JMS publisher creates and sends a message with the `JMSReplyTo` header set to some other destination defined within the JMS server. This can be a topic, queue, or a temporary topic or temporary queue. The Diffusion client does not have to know the destination type as this is handled within the adapter.

1. The Diffusion client subscribes to topic jms/queue/ABC.
2. The JMS provider creates a temporary queue, XYZ, and subscribes to it.
3. The JMS provider sends a message to the queue ABC with `JMSReplyTo` set to the queue XYZ.
4. The Diffusion client receives a message on queue jms/topic/ABC, with `DiffusionReplyTo` set to jms/reply/XYZ.
5. The Diffusion client sends a response message to the queue jms/reply/XYZ.
6. The JMS provider receives a TextMessage on the temporary queue XYZ.

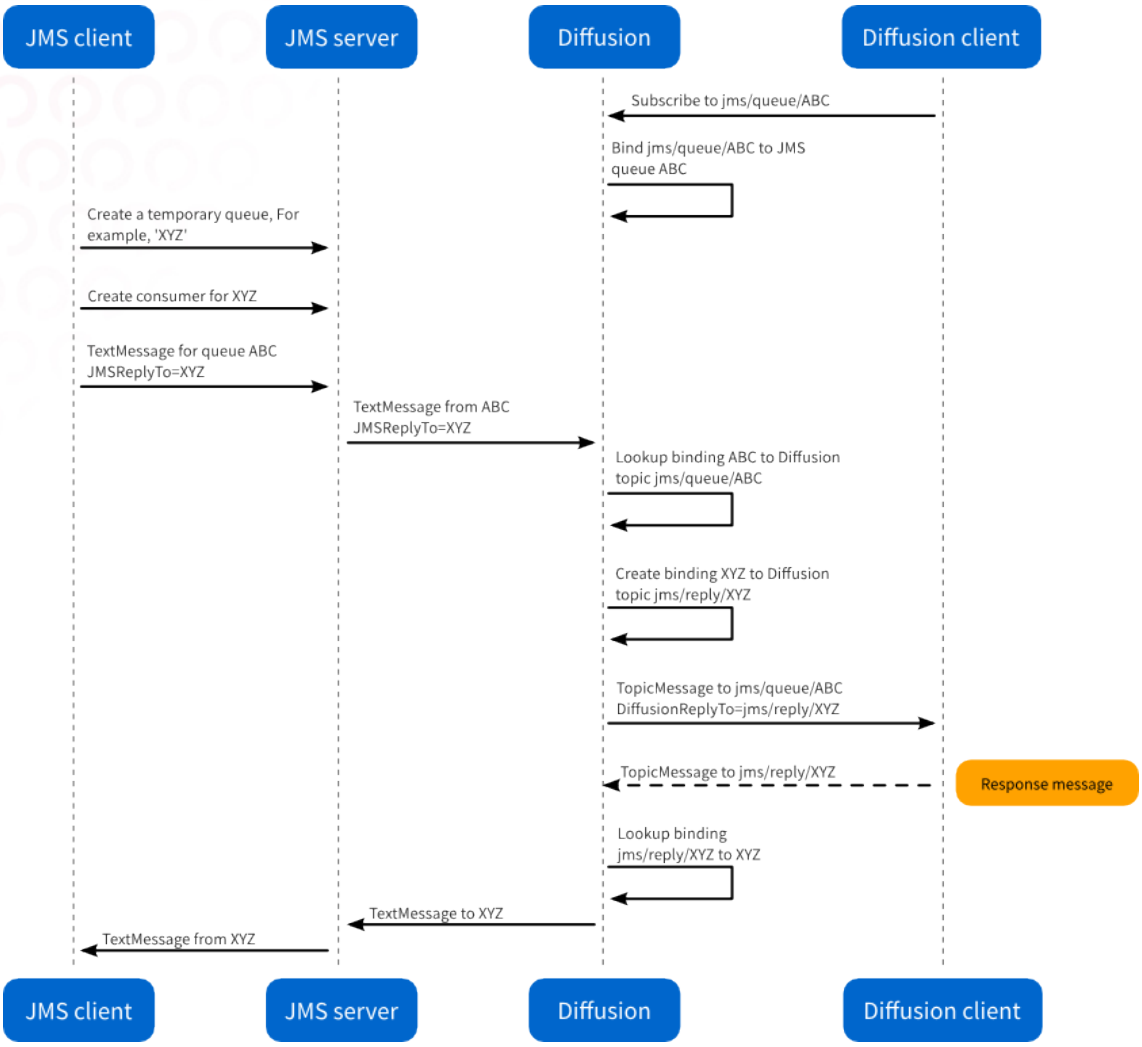


Figure 29: Request-reply initiated by a JMS client and serviced by a Diffusion client

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

Sending a request-reply message from a Diffusion client
You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

1. The JMS client subscribes to messages on the queue ABC.
2. The Diffusion client subscribes to jms/tmp/queue/XYZ. (Commonly, XYZ is a unique identifier).
3. The Diffusion client sends a request message to jms/queue/ABC with the DiffusionReplyTo header set with the value jms/tmp/queue/XYZ.

4. The JMS client receives the request message on queue ABC, with the JMSReplyTo header set to queue DEF.
5. The JMS client sends a reply to queue DEF.
6. The Diffusion client receives the reply on topic jms/tmp/queue/XYZ.

> **Note:** The return Diffusion topic can be any topic, so it is not necessary that the originating Diffusion client receives the reply – it can be any client listening for messages on that topic.

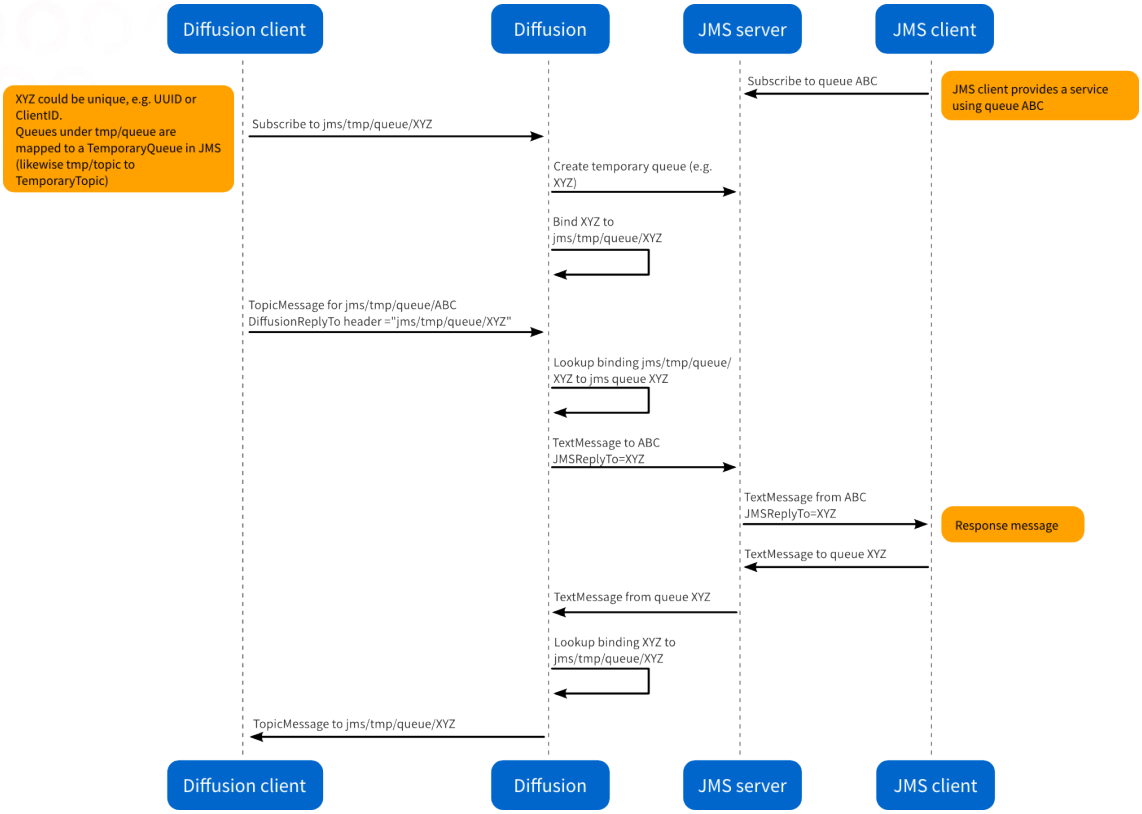

Diffusion client
Diffusion
JMS server
JMS client
XYZ could be unique, e.g. UUID or ClientID.
Queues under tmp/queue are mapped to a TemporaryQueue in JMS (likewise tmp/topic to TemporaryTopic)
Subscribe to queue ABC
JMS client provides a service using queue ABC
Subscribe to jms/tmp/queue/XYZ
Create temporary queue (e.g. XYZ)
Bind XYZ to jms/tmp/queue/XYZ
TopicMessage for jms/tmp/queue/ABC DiffusionReplyTo header ="jms/tmp/queue/XYZ"
Lookup binding jms/tmp/queue/XYZ to jms queue XYZ
TextMessage to ABC JMSReplyTo=XYZ
TextMessage from ABC JMSReplyTo=XYZ
Response message
TextMessage to queue XYZ
TextMessage from queue XYZ
Lookup binding XYZ to jms/tmp/queue/XYZ
TopicMessage to jms/tmp/queue/XYZ
Diffusion client
Diffusion
JMS server
JMS client

Figure 30: Request-reply initiated by a Diffusion client and serviced by a JMS client

Related Links

DEPRECATED: Configuring the legacy JMS adapter version 5.1 on page 597
Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Configuring the JMS Adapter v5.1 on page 598
The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

page 601

Example solutions

This section includes some example solutions that you can refer to when designing your own solution.

Example: Simple solution

This solution uses a firewall to restrict incoming traffic and a load balancer to balance the traffic between multiple Diffusion servers.

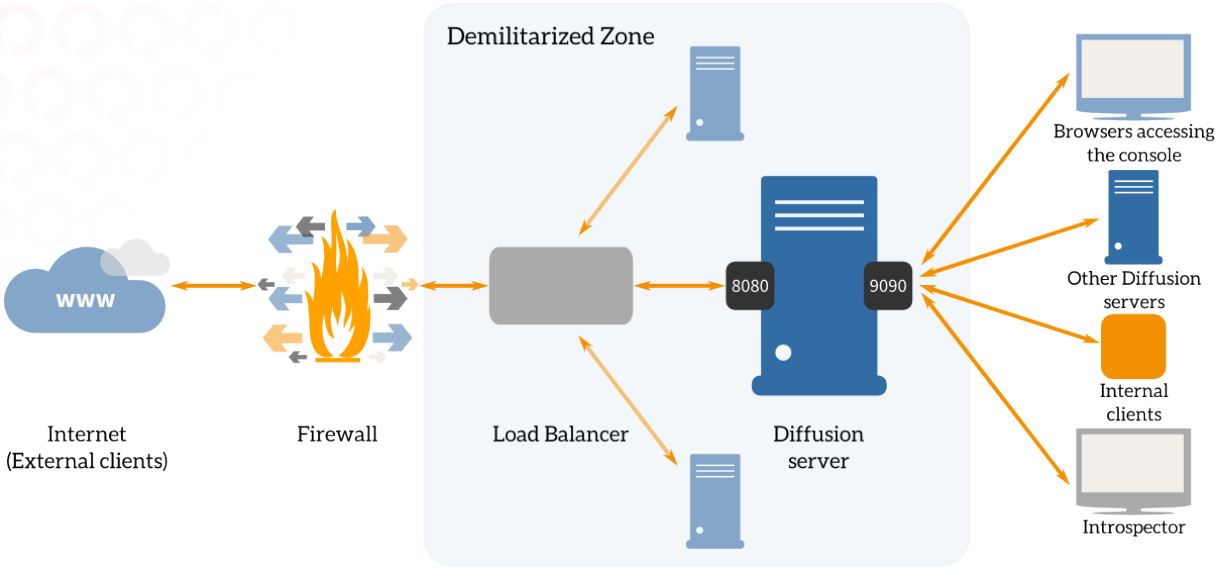


Figure 31: A simple solution

- Client applications can connect to Diffusion from the internet through a firewall.
- The firewall protects the DMZ from unwanted traffic. It allows connections on port 80 and redirects these connections to port 8080.
- The load balancer balances the Diffusion connections between all the Diffusion servers in the DMZ. You can also use the load balancer to filter the URL space and to perform SSL offloading.
- The Diffusion servers receive connections from external clients on port 8080. This port is protected by an authentication handler that performs strict authentication on the incoming connections. Authentication handlers can be local to the server or part of a control client.
- The Diffusion servers receive connections from internal clients on another port, for example 9090. The authentication controls on this port are less strict because these connections come from within your network. Internal connections can come from any of the following components:
  - Browsers accessing the Diffusion console
  - Other Diffusion servers that are not located in the DMZ.
  - Internal clients, such as control clients.
  - The Introspector

Example: Fan-out distribution

A fan-out distribution comprises many servers that host the same topic or topics. When the topic is updated on a primary server, the update is fanned out to the replica servers.

Fan-out using distributed topics

You can deploy publishers on replica servers that subscribe as clients to topics on the primary server. The publishers can use these updates to update topics on their host server.

Updating topics from delta updates received from a primary server is supported for the following types of topic data:

- Record topic data
- Single value topic data

To set up fan-out using distributed topics, you require the following components:

- On the primary Diffusion server
  - The topics that are to be replicated
  - A topic notify topic that updates when new topics are created and deleted on the primary server

  These topics can be created by a publisher or a control client.
- On one or more replica Diffusion servers
  - Publishers that subscribe to the topic notify topic and can choose to create slave topics and subscribe to the topics on the primary server. These publishers receive updates from the topics on the primary server and can apply them to the slave topics.

Figure 32: Fan-out using distributed topics

1. Publishers on the replica servers subscribe to a topic notify topic on the primary server.
2. A new topic is added to the primary server. This causes an update to the topic notify topic that notifies the publishers on the replica servers that the topic has been created.
3. Publishers on the replica servers can create a slave topic with the same name on their own server and subscribe to the topic on the primary server.
4. The topic is updated on the primary server. Updates are sent to the subscribing publishers. The publishers can use delta updates to update the slave topics on their server.

Fan-out using topic replication

You can use topic replication to synchronize topics across servers. Topic replication is set up on a topic branch basis in the server configuration. It duplicates replicated topics across all the servers with the same configuration.

Topic replication is primarily used for resilience and not suitable for large scale fan-out distributions.

Topic replication is supported only for the following types of topic data:

- No topic data
- Custom topic data
- Protocol buffer topic data
- Record topic data
- Single value topic data

For more information, see Topic replication on page 86.

Related Links

Distributed topics
Topic notify topic data
Publishing topic data

Example: A solution using clients

Clients with different uses connect to the Diffusion server in this example solution.

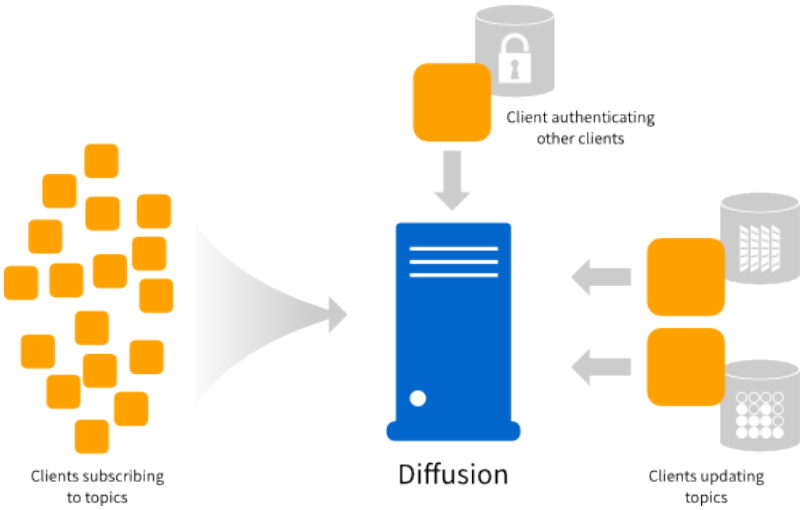


Figure 33: Clients for different purposes

This example solution uses three kinds of client, each for a different purpose:

Clients subscribing to topics

These clients are used by your customers to receive the data you distribute. You can use any of the provided APIs to create these, depending on how your customers want to access your data. For example,

- Use the iOS Classic API to create an iPhone app.
- Use the JavaScript Unified API to create a browser client.

These clients subscribe to the topics that are of interest to your customer, receive updates published on these topics, and display the information to your customers.

Clients creating and updating topics

These clients are used by your organization to distribute your data. You must use an API that provides control features to create these clients. For example, the JavaScript Unified API or the .NET Unified API.

These clients create the topics required to support your data structure and to publish data from your data sources to topics on the Diffusion server.

Clients authenticating other clients

These clients are used by your organization to authenticate connections from other clients. You must use an API that provides control features to create these clients. For example, the Java Unified API.

These clients are called by the Diffusion server to provide an authentication decision when another client connects to the Diffusion server anonymously or with a principal. In addition to deciding whether the other client is allowed to connect, the authenticating client can assign roles to the client session.

The authenticating client can use information stored elsewhere in your system, for example in an LDAP server, to make the authentication decision and assign roles.

Support

When designing your solution, refer to the support information to ensure compatibility between Diffusion and your hardware, software, and operating systems.

System requirements for the Diffusion server

Review this information before installing the Diffusion server.

The Diffusion server is certified on the system specifications listed here. In addition, the Diffusion server is supported on a further range of systems.

Certification

Push Technology classes a system as certified if the Diffusion server is fully functionally tested on that system.

We recommend that you use certified hardware, virtual machines, operating systems, and other software when setting up your Diffusion servers.

Support

In addition, Push Technology supports other systems that have not been certified.

Other hardware and virtualized systems are supported, but the performance of these systems can vary.

More recent versions of software and operating systems than those we certify are supported.

However, Push Technology can agree to support Diffusion on other systems. For more information, contact Push Technology.

Physical system

The Diffusion server is certified the following physical system specification:

- Intel™ Xeon™ E-Series Processors
- 8Gb RAM
- 8 CPUs
- 10 Gigabit NIC

NIC, CPU, and RAM (in decreasing order of importance) are the components that have the biggest impact on performance. High performance file system and disk are required. Intel hardware is used because of its ubiquity in the marketplace and proven reliability.

Virtualized system

The Diffusion server certified on the following virtualized system specification:

Host

- Intel Xeon E-Series Processors
- 32Gb RAM
- VMware vSphere® 5.5

Virtual machine

- 8 VCPUs
- 8Gb RAM

Operating system

Diffusion is certified on the following operating systems:

- Red Hat® 6.5, 6.6, and 7.1
- Windows™ Server 2012 R2

We recommend you install your Diffusion server on a Linux™-based operating system with enterprise-level support available, such as Red Hat Enterprise Linux.

Java

The Diffusion server is certified on Oracle Java 8 64-bit JDK

Only the Oracle JDK is certified.

The Diffusion server also runs on Java Development Kit 7. However, Oracle withdrew support for Java 7 in April 2015. We recommend that you use the latest update of Java 8.

Ensure that you use the Oracle JDK and not the JRE.

Networking

Push Technology recommends the following network configurations:

- 10 Gigabit network
- F5® load balancers with SSL offloading

Client requirements

For information about the supported client platforms, see Interfaces supported and Supported client platforms.

Platform support for the Diffusion libraries

Supported platforms and protocols for the client libraries

Table 15: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
JavaScript Classic API	V1.3 (es2)	• Native:WS, WSS, HTTP, HTTPS, HTTP Streaming • Flash: DPT, HTTP, HTTPS, HTTP Streaming • Silverlight: DPT, HTTP, HTTPS
JavaScript Unified API	es5	• WS, WSS
Java	8 (recommended), 7 (supported) **Note:** We recommend that you run your clients on the JDK rather than the JRE.	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)
.NET	4.5	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)
C	• Linux: Red Hat and CentOS™, version 6.5 and later (Tier 1 support) • Windows: Visual C Compiler 2013 (Tier 2 support) • OS X®: (Tier 2 support)	• DPT, DPTS
iOS Classic API	v5.1.1 device architectures: armv7, armv7s, arm64 simulator architectures: i386, x86_64	• DPT, DPTS • WS, WSS
Android	2.2	• DPT, DPTS
Flash	ActionScript V3.0 (Flex 3.0)	• DPT, DPTS • HTTP, HTTPS • HTTPC, HTTPCS
Silverlight	V4.0	• DPT

Protocols supported

Each client supports varying transports. A table of the supported transports for each client is presented here.

The following table describes a more detailed list of the implemented protocols. Some protocols have been partially implemented and are not supported.

The JavaScript client is fully supported only on certain browsers. Best effort support is provided for other browsers but the software/hardware combination might not be reproducible, particularly for mobile browsers. For more information about supported browsers, see Browsers supported on page 127.

The following table lists the protocols supported for each client:

Table 16: Supported protocols by client

Client	DPT	DPTS	WS	WSS	HTTP Full Duplex	HTTPS Full Duplex	HTTP Chunked Streamin	HTTPS Chunked Streamin	HTTP Polling	HTTPS Polling
JavaScript Unified API	✓ [3]	✓ [3]	✓	✓						
Java	✓	✓	✓	✓	✓	✓				
.NET	✓	✓	✓	✓	✓	✓				
JavaScript Classic API	✓ [3]	✓ [3]	✓	✓			✓ [4]	✓ [4]	✓	✓
Flash	✓	✓					✓	✓	✓	✓
Silverlight	✓	✓							✓	✓
iOS	✓	✓	✓	✓						
Android	✓	✓								
C	✓	✓								
Publisher client	✓ (Recommended)	✓	✓	✓	✓	✓				

[3] Supported by Flash/Silverlight
[4] Supported natively and by Flash

Browsers supported

Some of the client libraries are intended to be run within browser environments. Diffusion clients can use most commercial browsers and their variants. However, some Diffusion API protocols might not be available.

Diffusion supports the latest release of the following browser versions based on the original Diffusion release date.

Table 17: Supported browsers

Browser	Version
Google Chrome™	43
Mozilla Firefox®	38
Microsoft® Internet Explorer®	11.0
Apple Safari®	7.0.6

- Push Technology runs full regression tests on the browsers and versions documented above for every patch release.
- Push Technology carries out basic validation testing on the latest versions of these browsers but full support is available only at the next minor release.
- Support for older versions of browsers is provided on a best-effort basis, unless specified otherwise.
- Support for other browsers is provided on a best-effort basis.

Browser plugins (Flash and Silverlight)

Diffusion only supports official Flash and Silverlight players.

Table 18: Browser Plugins

Plugin player	Version
Adobe® Flash Player	18
Microsoft Silverlight	5.1

For details about the operating systems and browsers supported by the Silverlight plugin, refer to the “System Requirements” section on the following web page: Get Microsoft Silverlight

Browser limitations

Some browsers cannot use all the Diffusion protocols or features. If you experience problems when developing with protocols or client libraries that use the browser, check whether the browser supports this function.

Browser environments are not uniform and some browsers might have functional limitations. It is important to be aware of these limitations when developing for compliance with target browsers.

WebSocket limitations

WebSocket is an Internet Engineering Task Force (IETF) protocol used by Diffusion to provide a full-duplex communication channel over a single TCP connection. It is not supported by all browser versions.

- Internet Explorer
- Firefox
- Chrome
- Safari
- Opera®
- iOS
- Android

Table 19: Internet Explorer support for WebSocket

Version	WebSocket support?
7.0	NO
8.0	NO
9.0	NO
10.0	YES
11.0	YES (see note)

Note: Internet Explorer 11 contains a bug that causes WebSocket connections to be dropped after 30 seconds of inactivity. To work around this problem set the system ping frequency to 25 seconds or less. You can set the system ping frequency in the `Server.xml` configuration file. For more information, see Server.xml on page 622

Table 20: Firefox support for WebSocket

Version	WebSocket support?
34.0	YES
35.0	YES
36.0	YES
37.0	YES
38.0	YES

Table 21: Chrome support for WebSocket

Version	WebSocket support?
39.0	YES
40.0	YES
41.0	YES
42.0	YES
43.0	YES

Table 22: Safari support for WebSocket

Version	WebSocket support?
5.1	NO
6.1	YES
7.0	YES
7.1	YES
8.0	YES

Table 23: Opera support for WebSocket

Version	WebSocket support?
26.0	YES
27.0	YES
28.0	YES
29.0	YES
30.0	YES

Table 24: iOS support for WebSocket

Version	WebSocket support?
6.1	YES
7.0	YES
7.1	YES
8.0	YES
8.1	YES

Table 25: Android support for WebSocket

Version	WebSocket support?
4.2	NO
4.3	NO
4.4	YES
4.4.4	YES
40	YES

Cross-origin resource sharing limitations

CORS allows resources to be accessed by a web page from a different domain. Some browsers do not support this capability.

- Internet Explorer
- Firefox
- Chrome

- Safari
- Opera
- iOS
- Android

Table 26: Internet Explorer support for CORS

Version	CORS support?
7.0	NO
8.0	NO
9.0	NO
10.0	YES
11.0	YES

Table 27: Firefox support for CORS

Version	CORS support?
34.0	YES
35.0	YES
36.0	YES
37.0	YES
38.0	YES

Table 28: Chrome support for CORS

Version	CORS support?
39.0	YES
40.0	YES
41.0	YES
42.0	YES
43.0	YES

Table 29: Safari support for CORS

Version	CORS support?
6.0	YES
6.1	YES
7.0	YES
7.1	YES
8.0	YES

Table 30: Opera support for CORS

Version	CORS support?
26.0	YES
27.0	YES
28.0	YES
29.0	YES
30.0	YES

Table 31: iOS support for CORS

Version	CORS support?
7.0	YES
7.1	YES
8.0	YES
8.1	YES
8.3	YES

Table 32: Android support for CORS

Version	CORS support?
4.1	YES
4.3	YES
4.4	YES
4.4.4	YES
40	YES

Browser connection limitations

Browsers limit the number of HTTP connections with the same domain name. This restriction is defined in the HTTP specification (RFC2616). Most modern browsers allow six connections per domain. Most older browsers allow only two connections per domain.

The HTTP 1.1 protocol states that single-user clients should not maintain more than two connections with any server or proxy. This is the reason for browser limits. For more information, see RFC 2616 – Hypertext Transfer Protocol, section 8 – Connections.

Modern browsers are less restrictive than this, allowing a larger number of connections. The RFC does not specify how to prevent the limit being exceeded. Either connections can be blocked from opening or existing connections can be closed.

- Internet Explorer
- Firefox
- Chrome
- Safari
- Opera
- iOS

- Android

Table 33: Internet Explorer maximum supported connections

Version	Maximum connections
7.0	2
8.0	6
9.0	6
10.0	8
11.0	13

Table 34: Firefox maximum supported connections

Version	Maximum connections
34.0	6
35.0	6
36.0	6
37.0	6
38.0	6

Table 35: Chrome maximum supported connections

Version	Maximum connections
39.0	6
40.0	6
41.0	6
42.0	6
43.0	6

Table 36: Safari maximum supported connections

Version	Maximum connections
6.0	6
6.1	6
7.0	6
7.1	6
8.0	6

Table 37: Opera maximum supported connections

Version	Maximum connections
26.0	6

Version	Maximum connections
27.0	6
28.0	6
29.0	6
30.0	6

Table 38: iOS maximum supported connections

Version	Maximum connections
7.0	6
7.1	6
8.0	6
8.1	6
8.3	6

Table 39: Android maximum supported connections

Version	Maximum connections
4.2	6
4.3	6
4.4	6
4.4.4	6
40	6

Diffusion protocols like HTTP, HTTPS, HTTPC and HTTPCS use up to two simultaneous connections per Diffusion client. It is important to understand that the maximum number of connections is per browser and not per browser tab. Attempting to run multiple clients within the same browser might cause this limit to be reached.

Reconnection can be used to maintain a larger number of clients than is usually allowed. When TCP connections for HTTP requests are closed, the Diffusion sends another HTTP request which the server accepts. Be aware of cases where Diffusion tries to write a response to closed polling connections before the client can re-establish them. This behavior results in an IO Exception and the Diffusion server closes the client unless reconnection is enabled. When the client tries to re-establish the poll, it is aborted.

Another way to get around browser limits is by providing multiple subdomains. Each subdomain is allowed the maximum number of connections. By using numbered subdomains, a client can pick a random subdomain to connect to. Where the DNS server allows subdomains matching a pattern to be resolved as the same server, tab limits can be mitigated substantially.

Plugin limitations

When plugins such as Flash or Silverlight make HTTP requests they go through the browser instead of making them directly. This means they do not avoid browser limits on the number of connections and must wait for the browser to make the request. There is an additional layer between the client and the connection.

Browser buffering

When streaming, such as with HTTPC connections, the web browser might not release the data it receives immediately. Browsers might require that a minimum amount of data is received before passing the data on to the client. After this minimum amount has been received future messages are passed by the browser as soon as the browser receives them. This might cause problems when there is a delay between the initial topic load and delta messages.
To handle this the first message sent over a HTTPC poll is padded with null bytes to fill this buffer. The amount of padding sent can be configured with the comet-initial-message-padding configuration item. For more information, see WebServer.xml on page 642. The amount of padding required varies between browsers.

Security

Diffusion secures your data by requiring client sessions to authenticate and using role-based authorization to define the actions that a client can perform.

Concepts

Principal

The principal is a user or system user that has an identity that can be authenticated.

When a principal is authenticated is becomes associated with a session. The default principal that is associated with a session is ANONYMOUS.

Session

A session is a set of communications between the Diffusion server and a client.

Permission

A permission represents the right to perform an action on the Diffusion server or on data.

Role

A role is a named set of permissions and other roles. Principals and sessions can both be assigned roles.

Role hierarchy

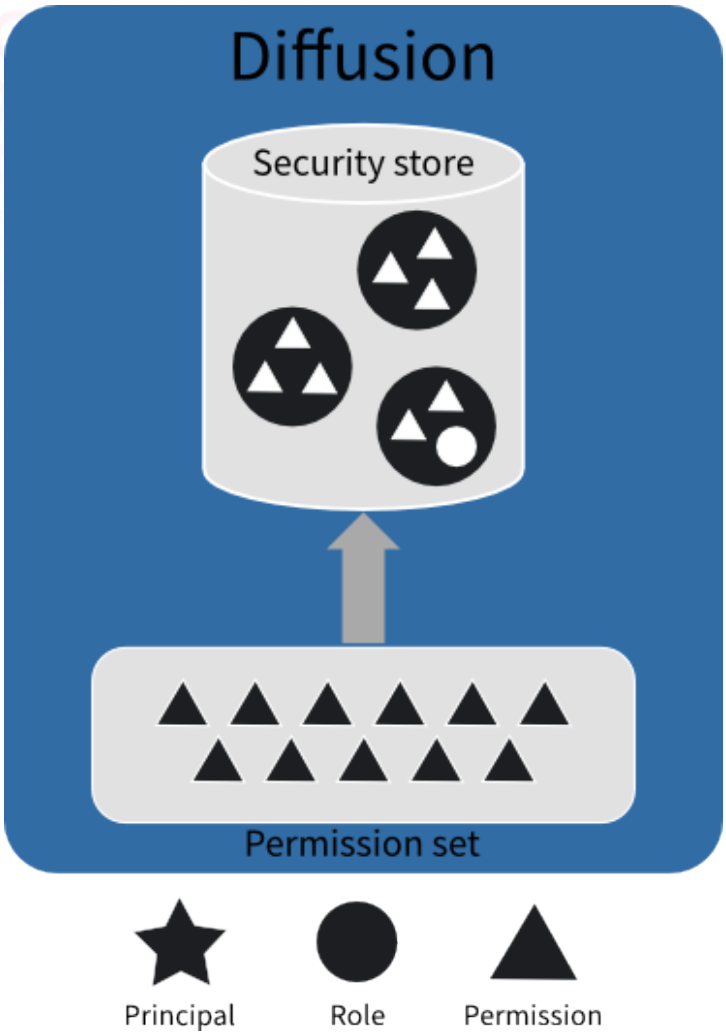
Roles are hierarchical. A role can include other roles and, by doing so, have the permissions assigned to the included roles. A role cannot include itself, either directly or indirectly – through a number of included roles.

Role-based authorization

Diffusion restricts the ability to perform actions to authorized principals. Roles are used to map permissions to principals.

Associating permissions with roles

The association between roles and permissions is defined in the security store.



A fixed set of permissions is defined by the Diffusion server. These permissions are used to control access to actions and data on the Diffusion server.

Roles are used to associate permissions to principals. Permissions are assigned to roles, and roles are assigned to principals.

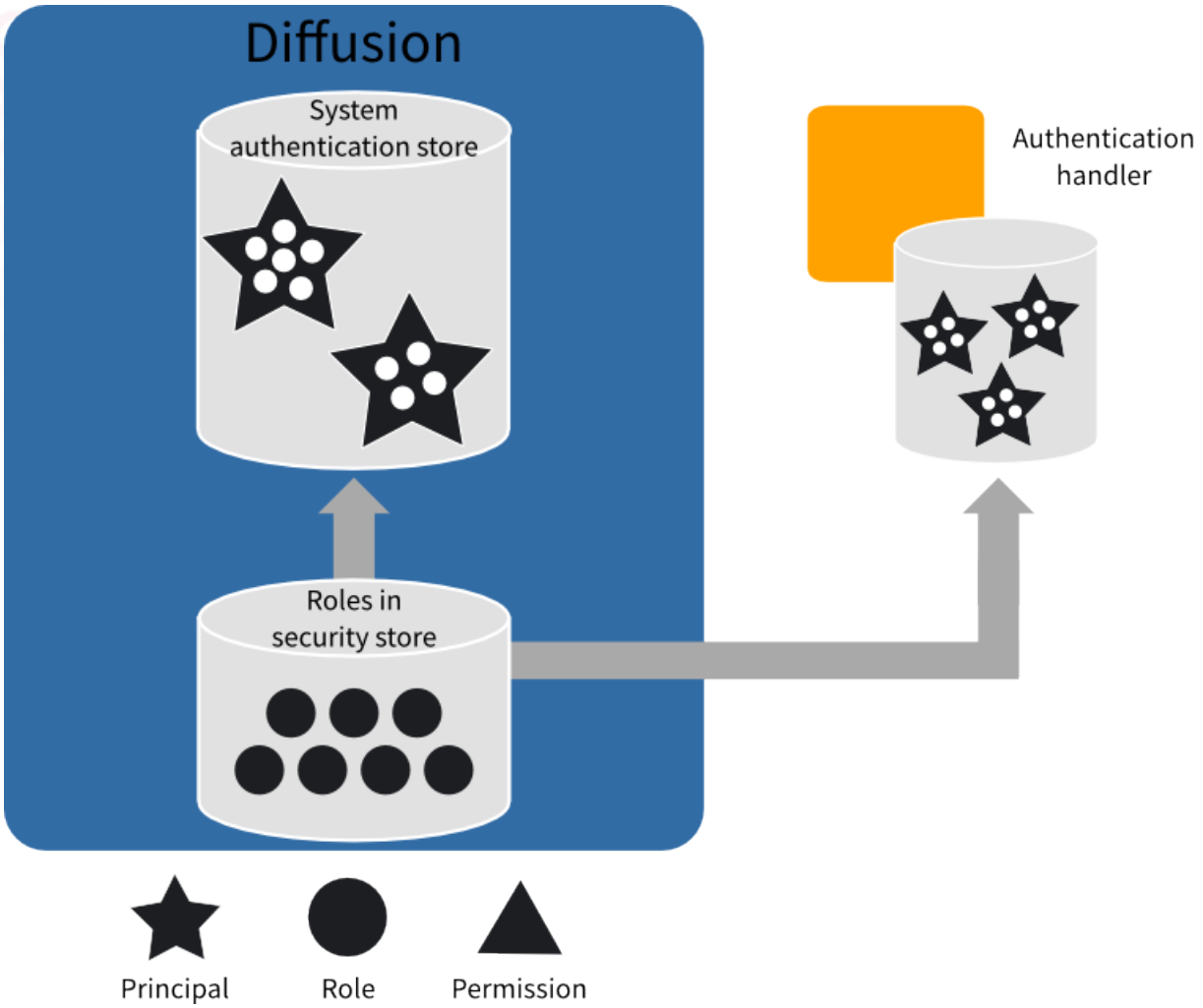
A role can be assigned zero, one, or many permissions. The same permission can be assigned to multiple roles. Roles can also include other roles to form a role hierarchy, and so inherit their permissions. The permissions assigned to a role and the role hierarchy are defined in the security store.

You can update the security store by editing the store file, *`installation_directory`*`/etc/Security.store`, and restarting the Diffusion server.

You can update the security store from a client using the SecurityControl feature.

Associating roles with principals

The association between roles and principals is defined in the system authentication store or by user-written authentication handlers.



The association between principals and roles is defined in the following ways:

- In a user-defined store that your user-written authentication handlers refer to. For example, an LDAP server.
- A user-written authentication handler can also hard code the relationship between principals and roles, if that is appropriate.
- In the system authentication store of the Diffusion server

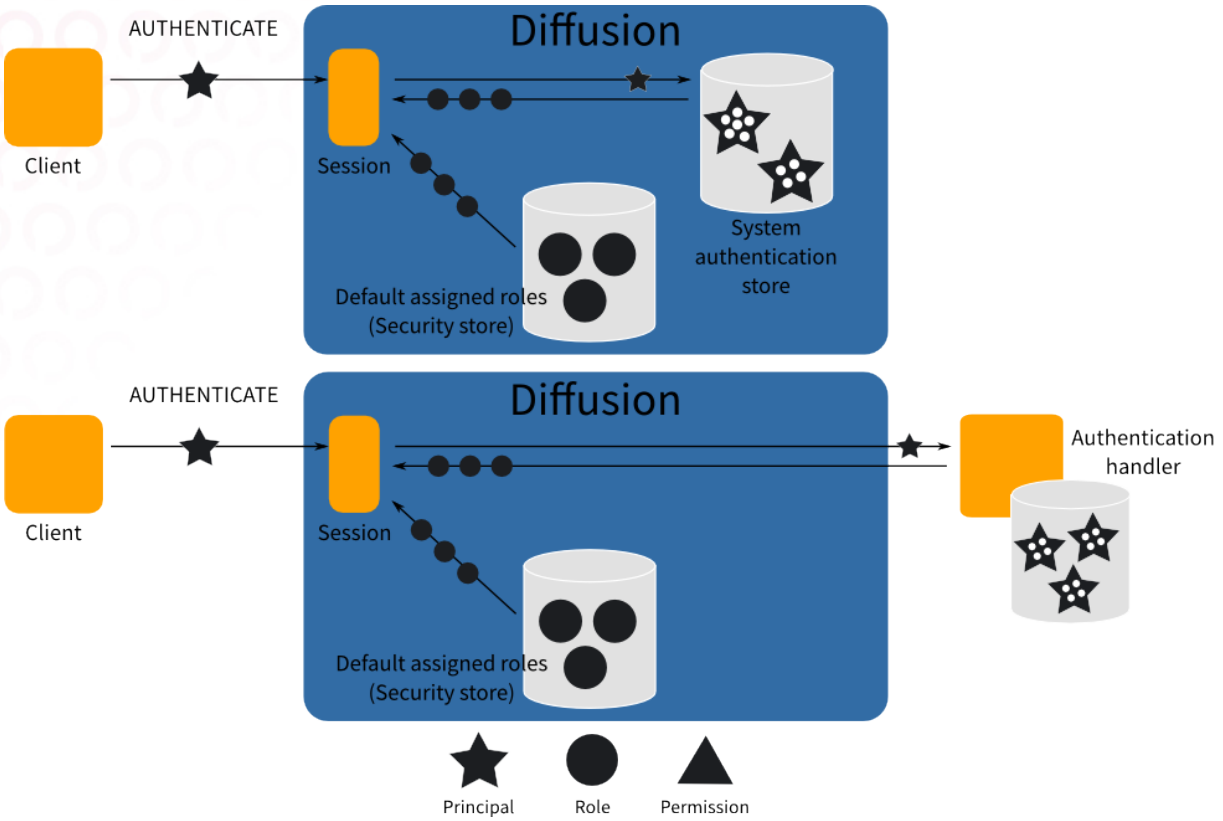
  The system authentication store is designed to hold information about Diffusion administration users and system clients. It can manage hundreds or perhaps thousands of principals, but does not provide the administration tools necessary to support millions of principals. We recommend that you delegate such "internet scale" use cases to a third-party identity provider using a custom authentication handler. For example, by using the OAuth or OpenID protocol.

You can update the system authentication store in the following ways:

- From a client using the SystemAuthenticationControl feature.
- By editing the store file, *`installation_directory`*`/etc/SystemAuthentication.store`, and restarting the Diffusion server.

Assigning roles to client sessions

Roles are assigned to a new client session after client authentication.



The roles assigned to a client session determine the actions that client session can perform.

A client session is assigned roles in the following ways:

- If the client session connects to the Diffusion server anonymously, the client session is assigned the default assigned roles for the ANONYMOUS principal.

  Anonymous authentication can be enabled or disabled in the system authentication store. If enabled, roles can also be specified.
- When a client session authenticates with a principal, the client session can be assigned the following roles:
  - The default assigned roles for a named principal.
  - The set of roles assigned to a principal by the authentication handler that accepts the client session's authentication request. This authentication handler can be one of the following types:
    - The system authentication handler, in which case the roles that are assigned are those associated with that principal in the system authentication store.
    - A user-written authentication handler, in which case the roles that are assigned are those defined by the handler or a user-defined store.

For example: A client session authenticates with the Diffusion server using the principal Armstrong. The first authentication handler that is called is a user-written authentication handler. This authentication handler abstains from the authentication decision, so does not assign roles to the client session. The next authentication handler that is called is the system authentication handler. The system authentication handler does not abstain from the authentication decision. It uses the information in the system authentication store to decide to allow the authentication request. In the system authentication store, the principal Armstrong is associated with the roles ALPHA, BETA, and EPSILON. These roles are assigned to the client session.

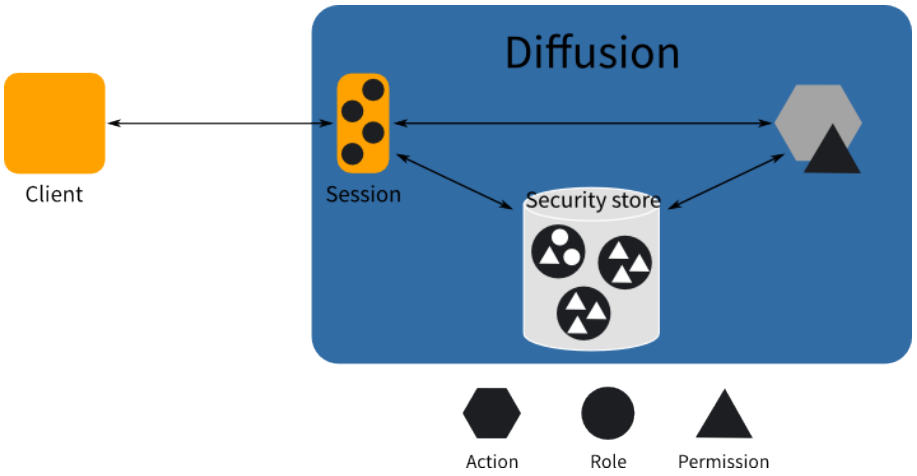
After the authentication request has been allowed, no further authentication handlers are called to make a decision or assign roles. However, the Diffusion server also assigns the default assigned roles for a named principal to the client session. The default assigned roles defined in the security store are GAMMA and RHO.

After authenticating with the principal Armstrong, the client session has the following roles assigned to it:

- ALPHA
- BETA
- EPSILON
- GAMMA
- RHO

Authorizing actions

When a client requests to perform an action or access data that requires a permission, the Diffusion server checks whether the client session is assigned a role that includes the required permission.



The client requests to perform an action. If the action requires that the client session have a permission, the Diffusion server checks what roles the client session is assigned and checks in the security store whether any of these roles have the required permission.

For example: A client requests to subscribe to the topic A/B/C. To subscribe to a topic, a client session must have the read_topic permission for that topic. The client session has the ALPHA and BETA roles. In the security store, the ALPHA roles does not include the read_topic permission, but the BETA roles does include the read_topic permission for the A/B/C topic. Because the client session is assigned the BETA role, it has the required permission and can subscribe to the topic.

Permissions

The actions a client session can take in Diffusion are controlled by a set of permissions. These permissions are assigned to roles.

Permissions can have one of the following scopes:

Topic

Permissions at topic scope apply to actions on a topic.

Topic-scoped permissions are defined against topic branches. The permissions that apply to a topic are the set of permissions defined at the most specific branch of the topic tree.

Global

Permissions at global scope apply to actions on the Diffusion server.

Topic permissions

The topic-scoped permissions are listed in the following table:

Table 40: List of topic-scoped permissions

Name	Description
read_topic	Subscribe or fetch topics at or below a topic branch.
update_topic	Update topics at or below a topic branch.
modify_topic	Create or modify topics at or below a topic branch.
send_to_message_handler	Send a topic message to the server for a topic at or below a topic branch.
send_to_session	Send a message to a client session for a topic at or below a topic branch.

Topic-scoped permissions are assigned to roles for specific topic paths. The permission assignment applies to all descendant topics, unless there is a more specific assignment.

To evaluate whether a client session has access to a permission for a topic, the Diffusion server starts at that topic and searches up the tree to find the nearest permission assignment. The first assignment is the only one considered, even if the client has roles involved in assignments further up the topic tree.

Default topic-scope assignments can also be defined. These are used if no path assignment matches.

The following example shows permissions set on branches of the topic tree.

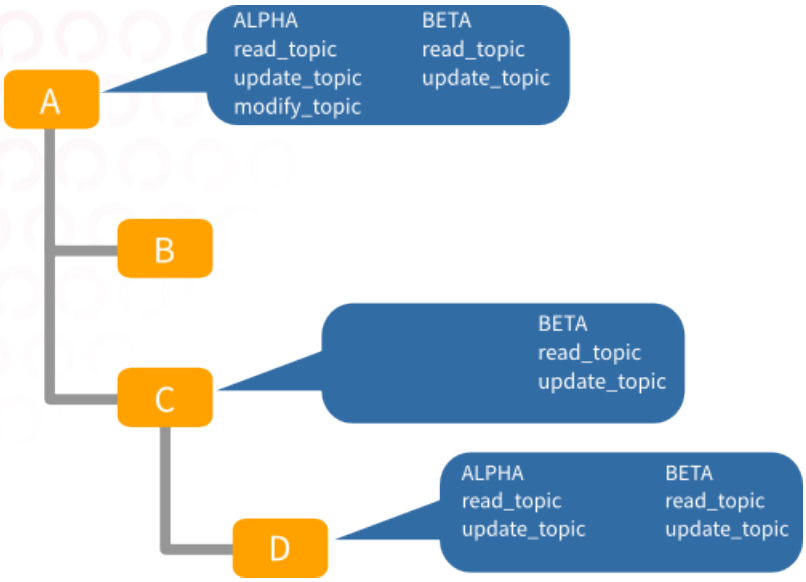


Figure 34: Topic scope example

In this example, client sessions with the role ALPHA have the following permissions on each topic in the topic tree:

A

A permission set is defined for the topic path A.

These permissions give client sessions with the ALPHA role read_topic, update_topic, and modify_topic permissions on the topic A.

A/B

No permission set is defined for the topic path A/B. In this case, the permissions at the most specific scope are those defined for the topic path A

These permissions give client sessions with the ALPHA role read_topic, update_topic, and modify_topic permissions on the topic B.

A/C

A permission set is defined for the topic path A/C. These permissions do not include any permissions for the ALPHA role. Client sessions with the ALPHA role have no permissions on the topic C.

Permissions are defined for the ALPHA role at a less specific scope. However, these permissions are not referred to or inherited if any permissions are defined at a more specific scope. Only the most specific set of permissions is used.

A/C/D

A permission set is defined for the topic path A/C/D.

These permissions give client sessions with the ALPHA role read_topic and update_topic permissions on the topic D.

The role ALPHA has only these permissions even though at A/C the role has no permissions defined and at A the role has additional permissions. Only the most specific set of permissions is used.

Global permissions

The global permissions are listed in the following table:

Table 41: List of global permissions

Name	Description
view_session	List or listen to client sessions.
modify_session	Alter a client session. This covers a range of actions including the following: • subscribe a session to a topic • throttle a session • enable conflation for a session • close a session
register_handler	Register any handler with the Diffusion server.
authenticate	Register an authentication handler. The register_handler permission is also required to perform this action.
view_server	Read administrative information about the Diffusion server. For example, through JMX.
control_server	• Shut down the Diffusion server. • Start and stop publishers. These actions can be taken only from the console or JMX. Client sessions cannot shut down the Diffusion server or start and stop publishers.
view_security	View the security policy.
modify_security	Change the security policy.
view_statistics	View statistics about the Diffusion server.

Pre-defined roles

Diffusion has a pre-defined set of roles with associated permissions.

You can use the SecurityControl feature to edit this set of roles.

AUTHENTICATION_HANDLER

- authenticate
- register_handler
- All permissions assigned to the “CLIENT” role.

ADMINISTRATOR

- view_server
- control_server
- view_security
- modify_security
- All permissions assigned to the “CLIENT_CONTROL” role.
- All permissions assigned to the “TOPIC_CONTROL” role.
- All permissions assigned to the “OPERATOR” role.

CLIENT_CONTROL

- view_session
- modify_session
- register_handler
- All permissions assigned to the “CLIENT” role.

TOPIC_CONTROL

- modify_topic
- send_to_session
- update_topic
- All permissions assigned to the “CLIENT” role.

CLIENT

- read_topic
- send_to_message_handler

OPERATOR

- view_server
- view_session
- read_topic on the topic Diffusion
- send_to_message_handler on the topic Diffusion
- All permissions assigned to the “CLIENT” role.

JMX_ADMINISTRATOR

- All permissions assigned to the “ADMINISTRATOR” role.

Authentication

You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

The handlers you can implement and register are the following:

- Any number of local authentication handlers
- Any number of control authentication handlers
- At most one authorization handler

Note: Using authorization handlers for authentication is deprecated.

The server calls the authentication handlers (local and control) in the order that they are defined in the `Server.xml` file. Then, if required, the server calls the authorization handler.

If no handlers are defined, the server allows the client operation by default.

Authentication process

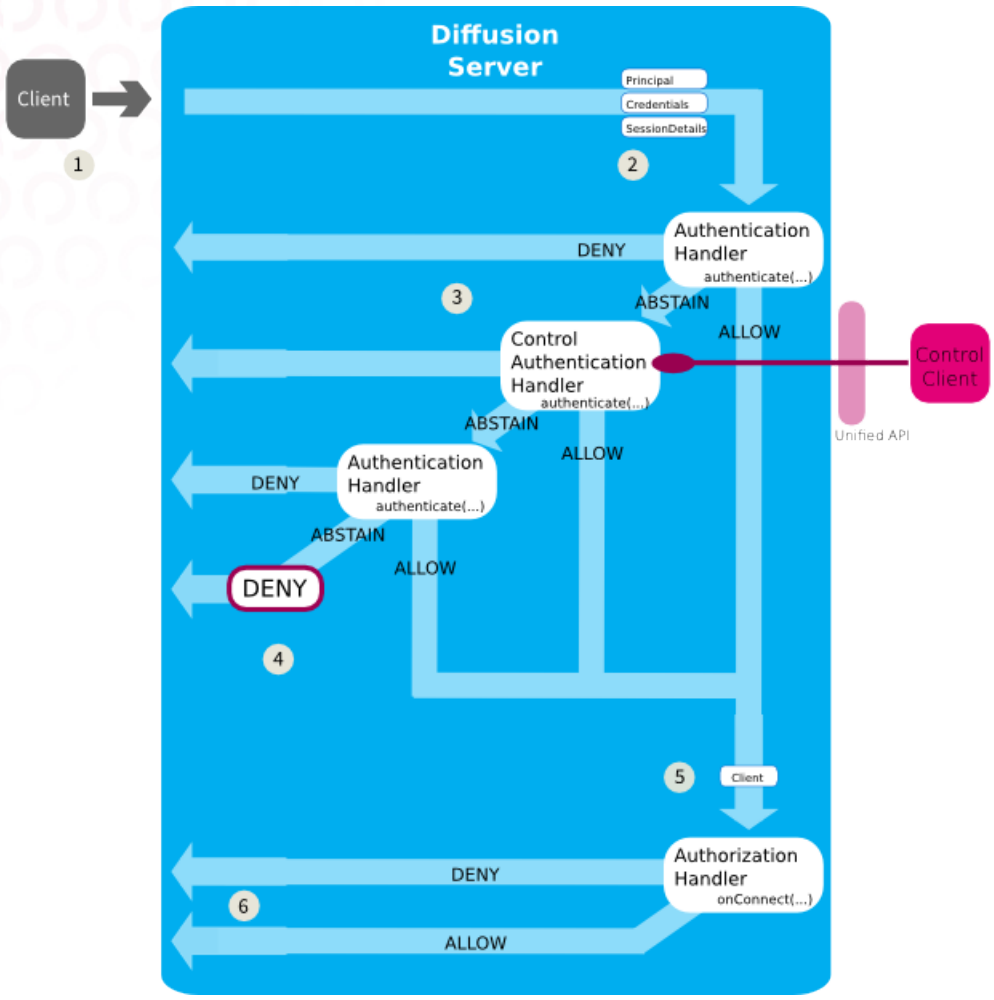


Figure 35: Authentication process for clients

1. A client tries to perform an operation that requires authentication. For more information, see Client operations that require authentication on page 144.
2. The server calls the authentication handlers one after another in the order that they are listed in the `Server.xml` file. It passes the following parameters to each authentication handler's `authenticate()` method:

   Principal

   A string that contains the name of the principal or identity that is connecting to the server or performing the action. This can have a value of `Session.ANONYMOUS`.

   Credentials

   The `Credentials` object contains an array of bytes that holds a piece of information that authenticates the principal. This can be empty or contain a password, a cryptographic key, an image, or any other piece of information. The authentication handler is responsible for interpreting the bytes.

   SessionDetails

   This contains information about the client. The available details depend on what information the server holds about the client session. Some session information might not be available on initial connection.

This information can be used in the authentication decision. For example, an authentication handler can allow connection only from clients that connect from a specific country.

When it registers with the server, a control authentication handler can specify what details it requires, so only these details are sent by the server (if available). This reduces the amount of data sent across the control client connection.

Callback

A callback that the authentication handler can use to respond to the authentication request by using the callback's `allow()`, `deny()`, or `abstain()` method.

If the authentication handler is a local authentication handler, the authentication logic is done on the server. If the authentication handler is a control authentication handler, the parameters are passed to a control client and the control client handles the authentication logic and returns a response.

3. Each authentication handler can return a response of `ALLOW`, `DENY`, or `ABSTAIN`.
   - If the authentication handler returns `DENY`, the client operation is rejected.
   - If the authentication handler returns `ALLOW`, the decision is passed to the authorization handlers. The authentication handler can also provide a list of roles to assign to the client session.
   - If the authentication handler returns `ABSTAIN`, the decision is passed to the next authentication handler listed in the `Server.xml` configuration file.
4. If all authentication handlers respond with an `ABSTAIN` decision, the response defaults to `DENY`.
5. If an authorization handler is configured in the `Server.xml` file, the server calls it. It passes the following parameter to the authorization handler's `onConnect()` method:
   Client

   The `Client` object has an associated `Credentials` object. This `Credentials` object is part of the Classic API and is different to the `Credentials` object used by the authentication handlers. The Classic API `Credentials` object contains two strings: username and password. The username string is equivalent to the Principal string used by the Unified API.
6. The authorization handler can return a response of `ALLOW` or `DENY`.
   - If the authorization handler returns `DENY`, the client operation is rejected.
   - If the authorization handler returns `ALLOW`, the client operation is allowed.

Client operations that require authentication

The following client operations require authentication with the server:

Table 42: Client operations that require authentication

Client operation	API version	Behavior if operation is allowed	Behavior if operation is denied
Connect to server	Unified API	The client connection to the server is accepted.	The client connection to the server is rejected and is dropped.

Client operation	API version	Behavior if operation is allowed	Behavior if operation is denied
Connect to server	Classic API	The client connection to the server is accepted.	The client connection to the server is rejected and is dropped.
Change the principal associated with a client session	Unified API	The principal is changed.	The principal is not changed, but the client session is not dropped.
Change the principal associated with a client session	Classic API	The principal is changed. In the Classic API, the principal is the username string inside the `Credentials` object.	The principal is not changed, but the client session is not dropped.

Related Links

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

User-written authentication handlers

You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

The authentication handlers can be implemented either remotely, in a client, or locally, on the server. The authentication handlers can be individual authentication handlers, that perform a single authentication check, or composite authentication handlers, that delegate to one or more individual authentication handlers.

Local authentication handlers

A local authentication handler is an implementation of the `AuthenticationHandler` interface. Local authentication handlers can be implemented only in Java. The class file that contains a local authentication handler must be located on the classpath of the Diffusion server.

For more information, see Authentication API.

Control authentication handlers

A control authentication handler can be implemented in any language where the Diffusion Unified API includes the AuthenticationControl feature. A control authentication handler can be registered by any client that has the authenticate and register_handler permissions.

For more information, see AuthenticationControl on page 300.

Composite authentication handlers

A composite authentication handler delegates the authentication decision to an ordered list of one or more individual authentication handlers and returns a combined decision.

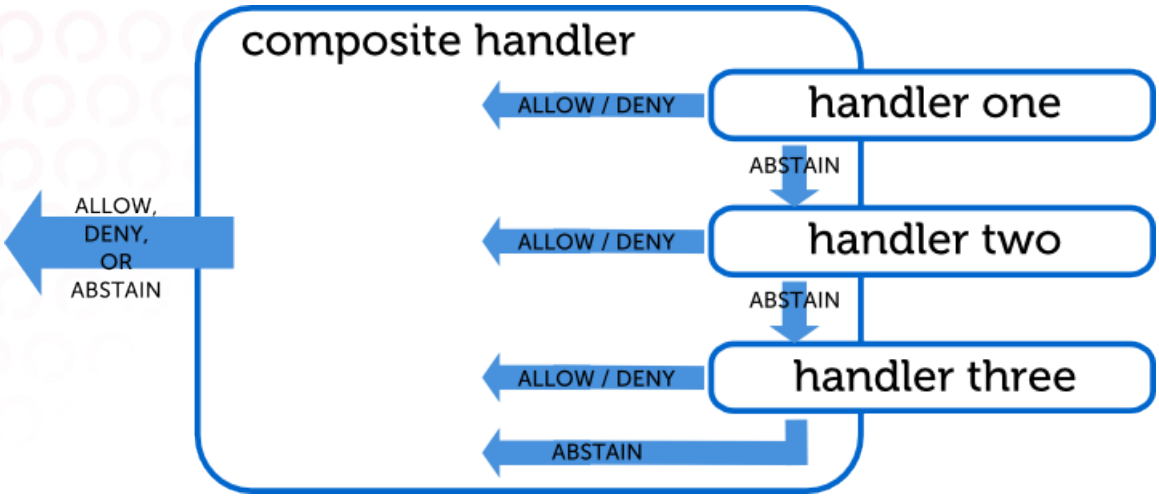


Figure 36: A composite authentication handler

- If an individual handler allows the client action, the composite handler responds with an ALLOW decision.
- If an individual handler denies the client action, the composite handler responds with a DENY decision.
- If an individual authentication handler abstains, the composite handler calls the next individual handler.
- If all individual handlers abstain, the composite handler responds with an ABSTAIN decision.

A composite authentication handler can be either local or control. A local composite authentication handler can delegate the authentication decision to one or more authentication handlers. A composite control authentication handler can delegate the authentication decision to one or more control authentication handlers.

The use of composite authentication handlers is optional. There are two reasons to consider using them:

- Composite authentication handlers enable you to combine authentication handlers together, which reduces the possibility of misconfiguration.
- Composite control authentication handlers improve efficiency by reducing the number of messages sent between the Diffusion server and clients.

The following table matrix shows the four types of authentication handler.

Table 43: Types of authentication handler

	Individual	**Composite**
Local	Implement the `AuthenticationHandler` interface. For more information, see Developing a local authentication handler.	Extend the `CompositeAuthenticationHandler` class. For more information, see Developing a composite authentication handler
Control	Implement the `ControlAuthenticationHandler` interface. For more information, see Developing a control authentication handler on page 307.	Extend the `CompositeControlAuthenticationHandler` class. For more information, see Developing a composite control authentication handler on page 310

Related Links

Authentication API

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

Server.xml on page 622
This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

Developing a local authentication handler

Developing a composite authentication handler

Developing a control authentication handler on page 307
Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

Developing a composite control authentication handler on page 310
Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

Server.xml on page 622
This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

Developing a local authentication handler

Developing a composite authentication handler

Developing a control authentication handler on page 307
Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

Developing a composite control authentication handler on page 310
Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

System authentication handler

Diffusion provides an authentication handler that uses principal, credential, and roles information stored in the Diffusion server to make its authentication decision.

System authentication store

The principal, credentials, and role information located in the system authentication store is used by the system authentication handler to authenticate users.

The system authentication store is designed to hold information about Diffusion administration users and system clients. It can manage hundreds or perhaps thousands of principals, but does not provide the administration tools necessary to support millions of principals. We recommend that you delegate such "internet scale" use cases to a third-party identity provider using a custom authentication handler. For example, by using the OAuth or OpenID protocol.

By default the following information is set in the system authentication store:

```
allow anonymous connections [ "CLIENT" ]
```

```
add principal "client" "password" [ "CLIENT" ]
add principal "control" "password" [ "CLIENT_CONTROL"
 "TOPIC_CONTROL" "AUTHENTICATION_HANDLER" ]
add principal "admin" "password" [ "ADMINISTRATOR" ]
add principal "operator" "password" [ "OPERATOR" ]
```

Behavior of the system authentication handler

The system authentication handler behaves in the following way:

- If anonymous connections are allowed in the system authentication store and a client session connects anonymously, the system authentication handler returns an ALLOW decision and the list of roles an anonymous client session is assigned.
- If anonymous connections are not allowed in the system authentication store and a client session connects anonymously, the system authentication handler returns a DENY decision.
- If a client session connects with a principal listed in the system authentication store and the correct credentials, the system authentication handler returns an ALLOW decision and the list of roles that client session is assigned.
- If a client session connects with a principal listed in the system authentication store and incorrect credentials, the system authentication handler returns a DENY decision.
- If a client session connects with a principal that is not listed in the system authentication store, the system authentication handler returns an ABSTAIN decision.

Related Links

Updating the system authentication store on page 313
A client can use the SystemAuthenticationControl feature to update the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users and assign roles to them.

SystemAuthenticationControl on page 312
Use the SystemAuthenticationControl feature to enable a client session to update the system authentication store. Principals and their associated credentials and roles are stored in the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users.

Authorization handlers

An authorization handler can control authorization and permissioning for clients and users.

Role-based authorization

> **Attention:** The new role-based security model has superseded authorization handlers. Role-based security enables you to more simply manage permissions and users. We recommend you use role-based authorization instead of authorization handlers. For more information, see Role-based authorization on page 134.

An authorization handler is a user-written Java class that must implement the `AuthorisationHandler` interface in the Classic API.

Such a handler can be used to restrict access of clients according to any criteria that is appropriate. One capability within Diffusion is for a client to be able to specify Credentials when they connect that can be checked by the authorization handler.

The handler can either be specified in `etc/Server.xml` in which case it is loaded when the server starts or can be set programmatically within a publisher using the `Publishers.setAuthorisationHandler` method.

There can only be one handler and it is system wide across all publishers, although you can have authorization at the publisher level.

If an authorization handler is not specified, credentials sent by a client are assumed to be valid. A publisher has access to the credentials to perform finer-grained authorization, if required.

The authorization handler interface has the following methods:

Table 44: Authorization handler methods

DEPRECATED: `canConnect(Client)`	This method is called to establish whether the client can connect and is called before any client validation policy is called.
`canSubscribe(Client, Topic)`	This method is called when a client subscribes to a topic. If topic information is sent with the connection, this method is called after the `canConnect` method. **Note:** This is called for every topic being subscribed to, even if subscribed as a result of a topic selector being specified. However (by default), if a topic is rejected by this method, it is not called again for any children (or descendants) of the topic.
`canSubscribe(Client, TopicSelector)`	This method is called when a client attempts to subscribe to a topic selector pattern (as opposed to a simple topic name). If topic information is sent with the connection, this method is called after the `canConnect` method.
`canFetch(Client, Topic)`	This method is called when a client sends a fetch request to obtain the current state of a topic. **Note:** This is called for every topic being fetched, even if fetched as a result of a topic selector being specified. However (by default), if a topic is rejected by this method, it is not be called again for any children (or descendants) of the topic.
`canFetch(Client, TopicSelector)`	This method is called when a client attempts to fetch topics using a topic selector pattern (as opposed to a simple topic name).
`canWrite(Client, Topic)`	This method is called when a client sends a message on a given topic, if false is returned the message is ignored, and the publisher will not be notified of the message. When implementing this method, be aware that performance can be impacted if many clients send messages or if a few clients send large messages.

DEPRECATED: `credentialsSupplied(Client, Credentials)`	This method is called when a client submits credentials after connection. It can be used to validate the credentials and must return true if the credentials are OK. If this returns false, a Credentials Rejected message are sent back to the client.

Authentication

> **Note:** The use of authorization handlers for authentication is deprecated. We recommend that you re-implement your authentication logic using authentication handlers. For more information, see User-written authentication handlers on page 145.

When a client connects to Diffusion it has the option of supplying user credentials. These credentials are basically tokens called username and password. These tokens can be used for any purpose. When `canConnect` is called, you can get the credentials from the `Client` object.

An example of this is:

```
   public boolean canConnect(Client client) {
Credentials creds = client.getCredentials();

// No creds supplied, so reject the connection
if (creds == null) {
    return false;
}

String username = creds.getUsername().toLowerCase();
```

If the credentials are null, none were supplied, which is different from empty credentials.

Clients can connect without credentials and submit them later or replace the credentials at any time whilst connected. The authorization handler is notified when new credentials are submitted and can choose to set the new credentials on the client.

The `Credentials` class has username and password attributes, but also allows for an attachment. It is here that a user normally sets any security object required. Returning true will allow the user to connect, returning false will result in the client connection being refused.

Subscription authorization

Subscription authorization is the allowing of a client to subscribe to a topic. In this case the `canSubscribe` is called. Returning true here allows the publisher to have any topic loaders and subscription methods called. Returning false will not notify the client that the subscription was invalid.

```
public boolean canSubscribe(Client client, Topic topic) {

    // Everyone has access to the top level topic
    if (topic.getName().equals(CHAT_TOPIC)) {
        return true;
    }

    User user = (User) client.getCredentials().attachment();

    return user.isAllowedRoom(topic.getNodeName());
}
```

Authorization handler

Authorization at the publisher level can also be achieved. This is required if there are many publishers running within the same Diffusion Server and they have different security settings. The following code example works if the publishers all implement `AuthorisationHandler`

```
public boolean canSubscribe(Client client, Topic topic) {

    AuthorisationHandler handler =

 (AuthorisationHandler)Publishers.getPublisherForTopic(topic);

    // Call the publisher in question
    return handler.canSubscribe(client, topic);
}
```

Permissioning

The permissioning process governs whether a client is able to send messages to a publisher, or in other words, is the topic read only. This is handled by the `canWrite` method. Again a good pattern might be to look at the credentials attachment object to see if this is permissible.

```
public boolean canWrite(Client client, Topic topic) {
 User user = (User) client.getClientCredentials().attachment();
 return user.canWriteMessages(topic);
}
```

Part III

Developer Guide

This guide describes how to develop publishers, client applications, and server-side components that interact with the Diffusion server.

In this section:

- Developing a publisher
- Developing a client
- Testing

Developing a publisher

You can develop a publisher in Java by using the Publisher API.

Writing a publisher

How to approach writing a publisher

> **Note:** This section covers only the main aspects of the publisher API. See the API documentation for full details.

There are demo publishers issued with Diffusion which have the source provided and these act as examples of working publishers.

In its simplest sense a publisher is responsible for providing topics, and publishing messages relating to those topics.

Before a publisher is written you need to carefully consider what it needs to do and what methods need to be implemented. The areas that need to be considered and the methods relating to them are discussed in the following sections.

There are many ways to approach the design of a publisher. For more information, see Design patterns.

Related Links

Classic deployment on page 663
Installing publishers into a stopped Diffusion instance.

Creating a Publisher class

A publisher is written by extending the abstract `Publisher` class (see Publisher API) and overriding any methods that must be implemented to achieve the functionality required by the publisher.

In all but the simplest of publishers it is likely that other classes must be written to perform the functionality required of the publisher.

The considerations of which methods must be overridden are discussed further within this section.

After the class is written and compiled, you can deploy it in the Diffusion server by specifying its details in `etc/Publishers.xml`

Publishers can also be deployed as a DAR file, sidestepping `etc/Publishers.xml`

See the section on testing for information about how to test the publisher.

Publisher startup

When a publisher is first loaded by the Diffusion server it can also be automatically started.

If not automatically started (or if it has been manually stopped), a publisher can be manually started by using the System Management interface. In either case the publisher processing goes through a number of startup steps. During these steps the `initialLoad` and `publisherStarted` methods are called and these methods can be implemented by the publisher to perform any initial processing like setting up the initial data state or adding initial topics.

Data state

A publisher typically holds some data which it updates according to any data update events it might receive.

The data held by the publisher is referred to as its state. To be more specific, it is the topics provided by the publisher that have data state. It is up to the publisher whether the data state is managed as a whole or on topic by topic basis.

It is the responsibility of the publisher to initialize its state and keep it updated as appropriate. Clients that subscribe to topics usually want to know the current state of the data relating to that topic and the publisher provides this as an initial topic load message. Clients are notified of all changes to that state by the publisher sending out delta messages.

A publisher typically has its own data model represented by classes written to support the data for the publisher. Ways in which such a data model can be managed are discussed in Designing your data model on page 48.

Initial state

A publisher's data typically has some initial state which can be updated during the life of the publisher. The state clearly must be set up before a client requires it but exactly when this is done is up to the publisher.

The state of the data as a whole can be set up when the publisher starts. This can be done in the `initialLoad` method where all topics required can be set up and the data loaded as appropriate.

Alternatively, the state of the data relating to a topic can be initialized when the topic is added, which is not necessarily when the publisher is started.

Another option is that the initial state is provided by a data feed as it connects (or is connected to). If data is provided by a server connection, the initial state can be set up when the server connection is notified to the publisher or more typically the server provides an initial topic load message.

Data integrity

The integrity of the data is also the responsibility of the publisher and care must be taken to ensure that all updating of data state is thread-safe. For example, it must be borne in mind that a client can request a load of current state (for example, by subscription) at the same time as the state is being updated.

> **Note:** The topic data feature automatically handles such data locking and in other cases topics might be locked as and when required.

Providing data state

If clients are to use the fetch facility to obtain the current state of topics, it will be necessary to consider the implementation of the `fetchForClient` method of the publisher.

Data inputs

For a publisher to be able to publish data to clients it must have a source for that data.

The data can be obtained from some type of feed, perhaps provided by some external API or it can be from some other application communicating using Diffusion protocols. This is entirely up to the publisher but Diffusion does offer some mechanisms.

Server connections

A publisher can receive its data from another publisher in a distributed environment as if it were a client of that publisher. To do this it is necessary for the publisher to connect to the server of the remote publisher and subscribe to its topics.

Connection to remote servers can be set up automatically for a publisher by configuring the connection in `etc/Publishers.xml`. A reference to such a server connection can be obtained by the publisher using the `getServerConnection` method or all connections van be obtained using `getServerConnections`.

Alternatively, a publisher can explicitly connect to remote servers using `addServerConnection`.

However the connection is made, the publisher is notified of the connection via the `serverConnected` method.

If a server connection is closed by the remote server or lost the publisher is notified through the `serverDisconnected` method.

Once a serverConnection is established the publisher must subscribe to topics on it and receive topic messages through the `messageFromServer` method. Typically, the first such method is a load message providing the initial state for the topic. Processing of the messages can be done within this method or alternatively the publisher can specify topic listeners to handle the messages on a per topic basis.

For more information, see Publisher clients on page 90.

Control clients

A publisher can receive input from a control client.

Control clients can use the TopicUpdateControl feature to publish messages to topics. Where such topics have topic data the topic state is automatically updated and deltas are published to subscribed clients. Where topics do not have topic data, published messages are forwarded to subscribed clients (that is, it is assumed that the control client maintains the data state).

Control clients can also send messages to specific clients and these are forwarded to the clients automatically.

Handling client subscriptions

Clients subscribe to topics provided by publishers and whenever this occurs the publisher is notified through its subscription method. The publisher can perform any processing it requires on subscription.

Topic loading

Typically, on subscription, the publisher provides the client with the current state of the data for the topic. It can do this by creating a new topic load message and populating it with a representation of the state. Rather than doing this every time a client subscribes it is generally more efficient for the publisher to create a topic load message only when the state changes and send this same message out to every client that subscribes.

This provision of the current state is known as the topic load. This can be done in one of the following ways:

Topic load in `subscription` method

If the topic has not already been loaded by a topic loader (see below), the loaded parameter of the subscription method is false. In this case, the normal action is for the publisher to send a topic load message to the client (passed as a parameter to subscription) through its send method.

Topic loaders

A topic loader is an object that implements the `TopicLoader` interface and can be used to perform any topic load processing that is required for one or more topics. Topic loaders can be declared for a `Publisher` using the `Publisher.addTopicLoader` method. This is typically done in the `initialLoad` processing and must be done before any topics that are loaded by the topic loader are added.

Using topic data

The use of topic data is recommended for topic data management and if it is in use then topic loading is fairly automatic and the default `subscription` method deals with it.

Hierarchic subscription

When a client subscribes to a topic the publisher can choose to subscribe the client to other topics or to subordinate topics. This can be done using the `Client.subscribe` methods.

A client itself can request subscription to a hierarchy of topics using topic selectors but this is an alternative method of handling hierarchies.

Publishing messages

Publishing a message means sending it to all clients subscribed to a topic. The message itself nominates the topic to which it relates.

The most common reason for publishing a message is to send an update (or delta) to the topic's data to all subscribed clients but messages can be published for any reason. In the case of an update, this is done in the method that became aware of the update.

A message for publishing can be created and populated by the publisher and then published using publishing methods on the topic or the publisher itself.

Exclusive messages

To send a message from a publisher to all clients subscribed to a topic except one single client, it can use the publishExclusiveMessage method. This might be appropriate if the message being published is a result of receiving a message from a client which you do not want to send back to that client.

Message priority

The priority at which a message is to be sent can be altered from the normal priority. For example, an urgent message can be sent with high priority causing it to go to the front of the client's queue.

Message acknowledgment

If acknowledgment of a message is required then it can be set as an acknowledged message. If any clients do not respond to a acknowledged message within a specified timeout, the publisher is notified on its `messageNotAcknowledged` method.

Topic locking

When TopicData is in use, all locking of the topic state is handled automatically. However, when the state of the topic is maintained in some other manner (for example as a topic attachment), it is the responsibility of the publisher application to handle locking.

About this task

The publisher must consider the issue of locking the topic whilst its state is changed and delta messages published.

By default, all topics have locking enabled which allows the publisher to lock and unlock the topic as required. When a client subscribes to a topic the subscription method of the publisher, normally sends the current state of the topic to the client. With locking enabled, the topic is locked for the duration of the subscription, which prevents other threads from acquiring the topic lock.

Threads that update the topic state and publish messages must also lock the topic for the duration of the update and publish. If this technique is not employed, a delta message might be sent to a client before the subscription method has the opportunity to send a topic load message. This can cause a failure at the client if topic aliasing is in use as aliasing relies upon the topic load message reaching the client first. Even if topic aliasing were not in use, the client application must be prepared for a delta arriving before the topic load.

Procedure

The following code example shows how a method responsible for updating the topic state and publishing a delta handles topic locking:

Results

```
    void updateMyData(String newValue) throws APIException {
        theTopic.lock();
        try {
            MyData data = (MyData)theTopic.attachment();
            data.setValue(newValue);
            TopicMessage message =
theTopic.createDeltaMessage();
            message.put(newValue);
            publishMessage(message);
        }
        finally {
            theTopic.unlock();
        }
    }
```

Using the try/finally technique ensures that whatever happens the code that locks the topic also unlocks it, which prevents a lock being left on indefinitely. As mentioned earlier the topic is automatically locked during subscription so the above code prevents an update from occurring whilst a subscription is in progress.

Handling clients

Interacting with clients from within a publisher

A publisher is notified when a client subscribes to one of its topics through the `subscription` method and when the client unsubscribes the `unsubscription` method is called.

A publisher can receive message from clients and send messages to clients (see below).

A client can request the state of any topic or topics at any time even if not subscribed to it. This is referred to as 'fetch' request. Such a request can routed to the publisher's `fetchForClient` method if a topic has no topic data.

Other than the above, a publisher is not normally notified of any other client activity. However a publisher can choose to receive client notifications using the `Publishers.addEventListener` method. Using client notifications, a publisher can even handle a fetch request for a topic that does not exist and return a reply (using `Client.sendFetchReply`) without the overhead of actually creating a topic.

A publisher can also choose to close or abort clients.

Sending and receiving client messages

In addition to publishing messages to all clients subscribed to a topic, you can send a message to only a single client using the `Client.send` method.

A client can also send messages to the publisher and these are received on the `messageFromClient` method which handles them accordingly. Only implement this method if messages are expected from clients. Alternatively the publisher specifies topic listeners to handle the messages on a per topic basis.

The 'client groups' facility allows messages to be sent to all clients in a group. There is also an 'exclusive message' facility which caters for sending to all but one client in a group.

Publisher closedown

A publisher is stopped and removed when the Diffusion server closes but can also be stopped and restarted, or stopped and removed by using the System Management interface.

However a publisher is stopped it always goes through a set of closedown steps, during which the `publisherStopping` and `publisherStopped` methods are called. A publisher can implement these methods if required to perform any special processing such as tidying up resources used.

Publisher removal

When a publisher is finally removed (either during server closedown or by using System Management), it cannot be restarted again within the same server instance. After removal the publisherRemoved method is called and this gives the publisher the opportunity to perform any final tidy up processing.

Stopping and restarting using System Management

By default, you cannot stop and restart a publisher using the System Management functions because in order for this to work the publisher must cater for the integrity of its state when this happens. As topics are also removed during stopping, the publisher must also be able to restore these topics if it were restarted.

If a publisher does want to cater for stop and restart using System Management, it must override the `isStoppable` method to return true. The publisher code must be able to recover topics and data state on restart.

Testing a publisher

There are various ways you can test your publishers after you have written them and deployed them on a Diffusion server instance.

The easiest way to perform some initial tests is to start it and try it out using some of the supplied testing tools. For example, you can start one or more instances of the client test tool, connect each to the test server and subscribe to the publisher's topic or topics. The initial topic load data is displayed and any messages sent as deltas are also displayed in each client. This tool can also be used to send messages to the publisher from the client.

Ultimately such tests are limited and you might want to develop Java tests which simulate clients using the Java external client API (or the Windows external client API).

Test as soon as possible with the actual clients that are going to be used. So, for example, you might want to develop browser clients using JavaScript, Flash or Silverlight.

It can help to diagnose problems with the publisher if it has diagnostic logging encoded within it. Such logging can be provided only at fine level and this logging level used only during testing.

Client groups

Clients that are connected can be managed in client groups allowing messages to be sent to groups of clients

Client groups are a convenient way of managing groups of clients with common attributes.

A publisher can create a client group and add clients to it. Messages can be sent to the group of clients rather than to individual clients.

When a client disconnects it is automatically removed from all client groups of which it is a member.

A client group belongs to the publisher that created it and can be used only from within that publisher. Group names are unique within the publisher only.

Client groups are a feature of the Java API using the `Publisher` interface.

Creating a client group

A publisher can create a new client group at any time using `createClientGroup`. When creating the group it is given a name which must be unique within the publisher. When a client group is created a reference to a `ClientGroup` object is returned.

Adding clients to a group

Clients can be added to a client group using the `addClient` method on a `ClientGroup` object.

Sending messages to clients in a group

A message can be sent to all clients in a client group using the `ClientGroup.send` method. To send to all but a specified client use `publishExclusiveMessage`.

Removing clients from a group

To remove clients from a client group, use the `ClientGroup.removeClient` method.

Removing a client group
To delete / remove a client group that is no longer required, use the `Publisher.removeClientGroup` method.

Other methods

All clients within a client group can be listed using `ClientGroup.getClients`.

You can check whether a particular client is already a member of a client group using `ClientGroup.containsClient`.

To enquire upon which client groups a particular client belongs to you can use `Publisher.getClientGroupMembership`.

To get a reference to a named client group from a publisher use `getClientGroup`.

Temporary client groups

Normally client groups have publisher scope, but you can create temporary groups using the `union` and `intersect` methods. These methods allow for the creation of client groups which are not managed by the client group manager. These temporary groups allow for the sending of messages which contain clients from different groups.

Client notifications

A publisher can opt to receive certain notifications regarding clients. It does this by adding a `ClientListener` which can be the publisher itself or any other object that implements the `ClientListener` interface.

A listener is added using the `Publishers.addEventListener` method.

All notifications are passed a reference to the client in question which can be interrogated for further information as required.

Notifications received on the `ClientListener` interface are as follows:

Table 45: Client listener notifications

`clientConnected`	This is called whenever a new client connects. It is not necessarily a client that is subscribing to one of the publisher's topics.
`clientResolved`	This is called when a newly connected client is resolved. A client's full geographical information is not necessarily available as soon as a client connects and so this method is called separately after the client has been resolved.
`clientSubscriptionInvalid`	This is called whenever a client attempts to subscribe to a topic that does not exist. This might be because the topic is not yet available and this gives a publisher the opportunity to create the topic and subscribe the client to it.
`clientFetchInvalid`	This is called whenever a client attempts to fetch a topic that does not exist. This gives the publisher the opportunity to respond to fetch request on a non-existent topic. A publisher can even reply to such a request without having to create a topic using the `sendFetchReply` method.

`clientSendInvalid`	This is called whenever a client attempts to send a message to a topic that does not exist, or to which the client is not subscribed. This enables a client to send a message to a topic and for that topic to be created and subscribed to on demand, or send data when a response is never expected.
`clientQueueThresholdReached`	This is called whenever a client's queue breaches an upper queue notification threshold or returns to a lower queue notification threshold. Parameters indicate which threshold has been reached and the threshold value.
`clientCredentials`	This is called whenever a client supplies new credentials after connection. It is called after the authentication handlers and authorization handlers (if any exist) have validated the credentials.
`clientClosed`	This is called whenever a client disconnects. The reason for disconnection can be obtained using `theClient.getCloseReason` method.

Adding a ClientListener

You can add a `ClientListener` to listen for client notifications.

About this task

Procedure

So a publisher can add itself as a listener for client notifications as follows:

Results

```
public class MyPublisher extends Publisher implements
 ClientListener {

    protected void initialLoad() throws APIException {
        Publishers.addEventListener(this);
    }
```

Using DefaultClientListener

How to use the default client listener to avoid implementing all methods.

About this task

The publisher must implement all of the `ClientListener` methods.

Procedure

For convenience, an abstract `DefaultClientListener` class is provided which has empty implementations of all methods. This can be extended to produce a class which implements only the methods you are interested in. Alternatively an anonymous class can be used within the publisher as follows:

Results

```
    protected void initialLoad() throws APIException {
```

```
        Publishers.addEventListener(
            new DefaultClientListener() {
                public void clientConnected(Client client) {
                    LOG.info("Client {} connected",client);
                }

                public void clientClosed(Client client) {
                    LOG.info("Client {} closed",client);
                }
            });
    }
```

Developing a client

Unified API provides APIs for a number of platforms. Use these APIs to develop your Diffusion clients.

Unified or Classic API?

The provided Diffusion APIS can be classified as Unified or Classic.

Unified API

The Unified API is a consistent and modular API that provides an asynchronous and session-oriented approach to developing your clients.

The Unified API is the future direction of the Unified API. All new features will be developed in the Unified API.

Classic API

The Classic API was provided with previous versions of Diffusion. It will continue to be available for backwards compatibility and to provide support for additional platforms.

However, we recommend that you use the Unified API where it is available for your required platform.

Unified API

The Unified API available in version 5.0 and later provides a different approach to that of the Classic API.

Consistent

The Unified API provides a single consistent interface to communicate with the Diffusion server, whatever the role of your client application. In the Classic API, separate interfaces were provided for the different client roles: standard client, control client, and event publisher.

Modular

Interfaces are provided on a feature-by-feature basis. There is a clear delineation between features. At runtime, the client starts only those services that it uses. Later versions of Diffusion can provide additional features without requiring that you recompile your existing clients.

By contrast, the Classic API had a monolithic structure.

Asynchronous

All calls in the Unified API are asynchronous. Asynchronous calls remove the possibility of your client becoming blocked on a call. The Unified API also provides context-specific callbacks, enabling you to pass contextual information with a callback, and a wider range of event notifications than was available in the Classic API.

Independent

The Unified API is provided in library files independent of a Diffusion server installation. This enables you to develop against the interfaces without requiring a Diffusion server and license on your development system. The Classic API is available only as part of the Diffusion jar file.

Session-oriented

The Unified API introduces the concept of a session. A session can survive the temporary loss of connection to the server. Clients are notified when the connection is lost.

Future releases of Diffusion will automate reconnection, queue, and replay options, and will increase the resilience of sessions.

Key concepts in the Unified API

The following section describes some of the key concepts associated with the Unified API.

When designing your clients, consider these aspects of the Diffusion APIs.

Sessions

A session represents a logical context between a client and one or more Diffusion servers.

Typically, a session represents a single connection to a single server. However, in the event of connection failure the session can automatically reconnect to the same server or even to failover to another server and still retain its context. For more information, see Session replication

You can open a session by using a session factory and specifying the URL of the Diffusion server. Diffusion URLs take the following form:

```
scheme://host:port
```

scheme

The transport used to make the connection. For example, `dpt`, `dpts`, `http`, `https`, `ws`, or `wss`.

host

The host name or IP address of the system on which the Diffusion server is located.

port

The port on which the Diffusion server accepts connections from clients using the Unified API. You can configure which ports to provide connectors for in the `Connectors.xml` configuration file of the Diffusion server.

The act of opening the session establishes a connection with the server.

The session does not receive input from the server until it is started, but can be used to obtain features and perform certain setup actions before it is started.

You can use the session factory to specify certain session attributes, including the following:
Error handler

A callback for session errors. You must implement this callback to act upon session errors. Errors that can be handled by the application are typically returned through feature callbacks. The normal action for a session error is to close the session.

Listener

An optional callback that can be used to notify the client of state changes on the session.

Principal

The security principal. By default, a session uses a zero length string, "", as its principal. This value is interpreted as "anonymous".

Credentials

The credentials associated with the principal that enable the server to authenticate the principal.

Buffer sizes

The input and output buffer sizes. To gain optimum throughput, configure the client input buffer size to match the server output buffer size and configure the client output buffer size to match the server input buffer size.

SSL context

The SSL context if a secure connection is required.

When a session is opened it is assigned a unique session identifier by the server which identifies the session even if it becomes connected to another server.

The following diagram shows the session state model:

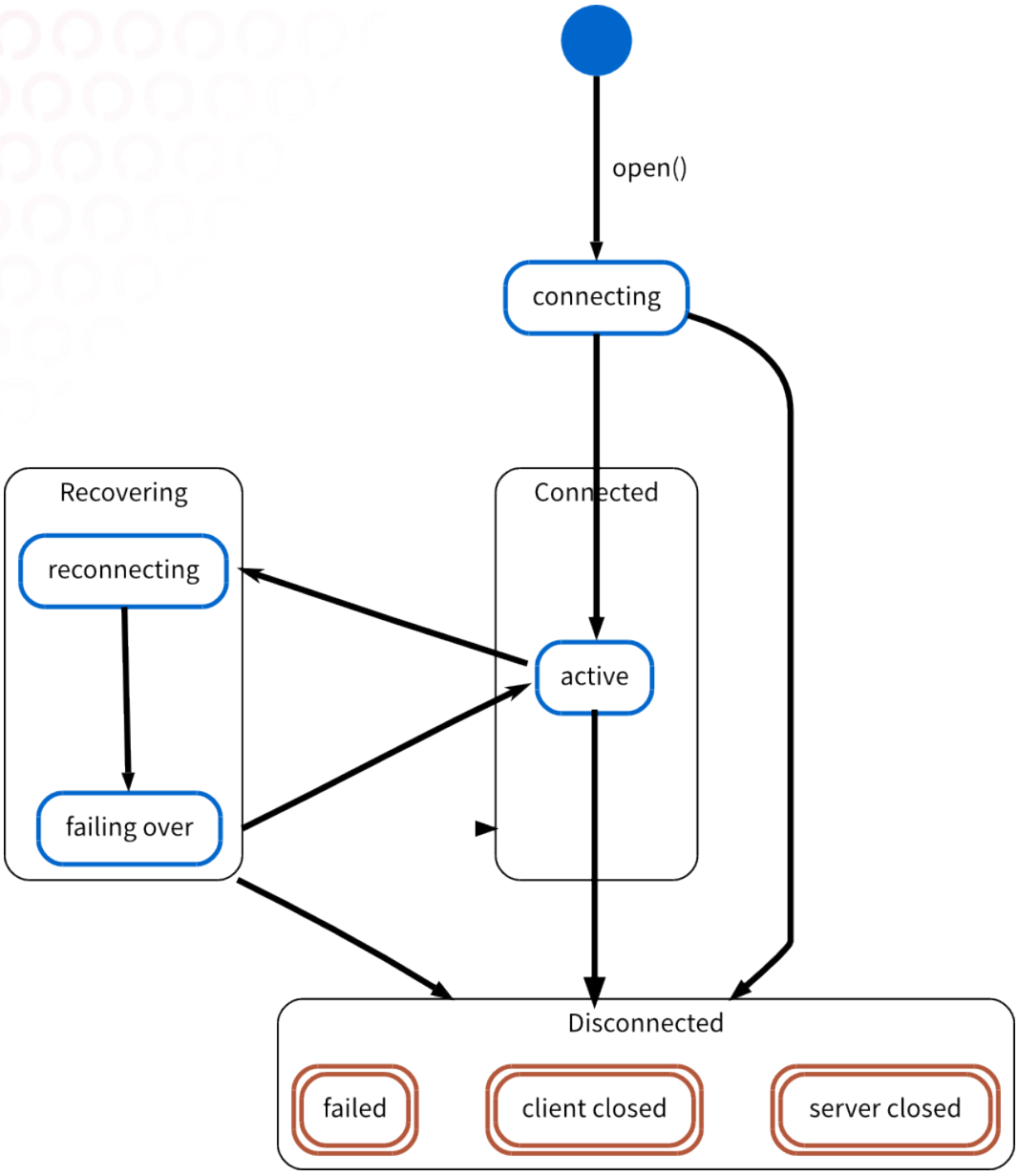


Figure 37: Session state model

Session properties

A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

Note: Session properties are not currently replicated through session replication.

Fixed properties

Fixed properties are set by the Diffusion server when a client opens a session with it. Fixed property keys are prefixed by a dollar sign ($). The fixed session properties are:

$SessionId

The session identifier.

$Principal

The security principal the session uses to connect to the Diffusion server.

$ClientType

The client type of the session. For more information, see Client types on page 95.

$Transport

The transport the client session uses to connect to the Diffusion server. For more information, see Client types on page 95.

$Connector

The name of the connector on which the client connected. the Diffusion server.

$Country

The two letter country code for the country where the client's internet address is located. The value is uppercase.

$Language

The two letter language code for the most common language of the country where the client's internet address is located. The value is lowercase.

User-defined properties

An authentication handler that allows the client session to connect can assign additional properties to the session. The keys of these properties are case sensitive, must begin with an alphabetic character, must be alphanumeric, and must not include any whitespace.

Related Links

Example: Use session property filters with messaging on page 295
The following examples use the MessagingControl feature in the Unified API to request session properties with messages sent to topic paths and to send messages to one or more clients depending on the values of their session properties.

Messaging to clients on page 289
A client can use the MessagingControl feature to send individual messages to any known client on any topic path. It can also register a handler for messages sent from clients.

Session filtering

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

To perform an action on a subset of the connected client sessions, you can create a query expression that filters the set of connected client sessions by the values of their session properties. Filter query expressions are parsed and evaluated by the Diffusion server.

The query expression used to filter the session is made up of one or more search clauses chained together by boolean operators.

Creating a single search clause

Search clauses have the following form:

```
key operator 'value'
```

key

The key name of the session property to be tested. The key name is case sensitive.

operator

The operator that defines the test to be performed. The operator is not case sensitive.

value

The test value to be compared to the session property value. This value is a string and must be contained in single or double quotation marks. Any special characters must be escaped with Java escaping. The value is case sensitive.

Table 46: Session filter search clause operators

Operator	Description
IS	Tests whether the session property value associated with the property key matches the test value.
EQ	Equals. Tests whether the session property value associated with the property key matches the test value. Equivalent to 'IS'.
NE	Not equal. Tests whether the session property value associated with the key is not equal to the test value.

Examples: single search clause

Filter by clients that connect with the principal Ellington:

```
$Principal IS 'Ellington'
```

Filter by clients that connect to the Diffusion server using WebSockets:

```
$Transport EQ 'WEBSOCKET'
```

Filter by clients that are not located in the United Kingdom:

```
$Country NE 'GB'
```

Filter by clients that have the user-defined property Location set to San Jose:

```
Location IS "San Jose"
```

Filter by clients that have the user-defined property Status set to Active:

```
Status EQ 'Active'
```

Filter by clients that do not have the user-defined property Tier set to Premium:

```
Tier NE 'Premium'
```

Chaining multiple search clauses

Chain individual search clauses together using boolean operator or use the NOT operator to negate a search clause. Boolean operators are not case sensitive.

Table 47: Session filter boolean operators

Operator	Description
AND	Specifies that both joined search clauses must be true.
OR	Specifies that at least one of the joined search clauses must be true.
NOT	Specifies that the following search clause or set of search clauses must not be true.

Use parentheses to group sets of search clauses and indicate the order of precedence for evaluation. If no order of precendence is explicitly defined, the AND operator takes precedence over the OR operator.

Examples: multiple search clauses

Filter by clients that connect with one of the principals Fitzgerald, Gillespie, or Hancock:

```
$Principal IS 'Fitzgerald' OR $Principal IS 'Gillespie' OR
 $Principal IS 'Hancock'
```

Filter by clients that connect to the Diffusion server using WebSockets and are located in France and have the user-defined property Status set to Active:

```
$Transport EQ 'WEBSOCKET' AND $Country IS 'FR' AND Status EQ
 'Active'
```

Filter by clients that are located in the United States, but do not connect with either of the principals Monk or Peterson:

```
$Country EQ 'US' AND NOT ($Principal IS 'Monk' OR $Principal IS
 'Peterson')
```

Filter by clients excluding those that have both the user-defined property Status set to Inactive and the user-defined property Tier set to Free:

```
NOT (Status IS 'Inactive' AND Tier IS 'Free')
```

Related Links

Example: Use session property filters with messaging on page 295
The following examples use the MessagingControl feature in the Unified API to request session properties with messages sent to topic paths and to send messages to one or more clients depending on the values of their session properties.

Messaging to clients on page 289

A client can use the MessagingControl feature to send individual messages to any known client on any topic path. It can also register a handler for messages sent from clients.

Features

Units of functionality within the Unified API are exposed as features.

You can obtain features from the session. You can have only one instance of each available feature per session. Each feature is dynamically instantiated only when it is first requested from the session. It can be efficient for a session to obtain references to the features it uses before starting the session.

Standard features are available to all sessions and enable clients to subscribe to topics to receive updates or to send and receive messages or both.

Control features are for control clients only and enable the client to perform actions upon the server similar to those that can be achieved from within an internal publisher. For example, topics can be created, updated, and deleted. A control client can also monitor and manage other client sessions.

Related Links

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

ClientControl on page 339
Use the ClientControl feature to enable a client session to receive notifications about other clients and to manage other clients.

SystemAuthenticationControl on page 312
Use the SystemAuthenticationControl feature to enable a client session to update the system authentication store. Principals and their associated credentials and roles are stored in the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users.

MessagingControl on page 288
Use the MessagingControl feature to enable a client to send messages to specific client sessions and receive messages sent by clients to topic paths.

Messaging on page 202
Use the Messaging feature to enable a client session to send messages to a topic path.

Pings on page 209
Use the Pings feature to enable a client session to ping the server and verify the connection between client and server.

TopicControl on page 219
Use the TopicControl feature to enable a client session to create and manage topics.

TopicUpdateControl on page 259
Use the TopicUpdateControl feature to enable a client session to update topics.

Topics on page 185
Use the Topics feature to enable a client session to subscribe to topics and receive streaming updates to those topics or to fetch the current state of topics.

Security on page 213
Use the Security feature to enable a client to change the principal associated with its session.

SubscriptionControl on page 344

Use the SubscriptionControl feature to enable a client to subscribe other clients to topics and handle routing topic subscription requests.

Callbacks and streams

The Unified API is asynchronous and uses callbacks to notify the client of successful or failed calls. The following types of callback exist: callback, stream, contextual callback, and contextual stream.

Callbacks

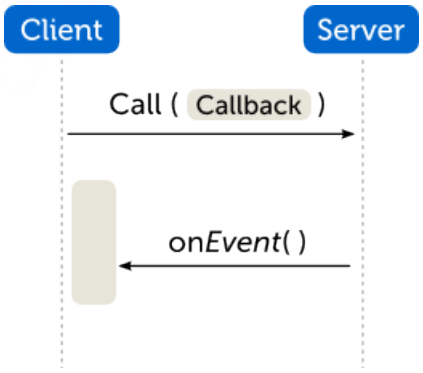


Figure 38: A callback

A callback is a response object that the client passes to an asynchronous call. When the call completes its actions it can use the callback object to respond to the client by calling a method on the callback. A callback can receive only one response from a call.

All callbacks in the Unified API have the following methods:

on*Event*

Where *Event* depends on the specific callback and describes the event that occurs when the call has completed. This method is used to provide the response from the call.

`onDiscard`

This method is used to indicate that the call has not returned a response, but that the callback is no longer required. This can happen when a call times out or a session is closed.

Streams

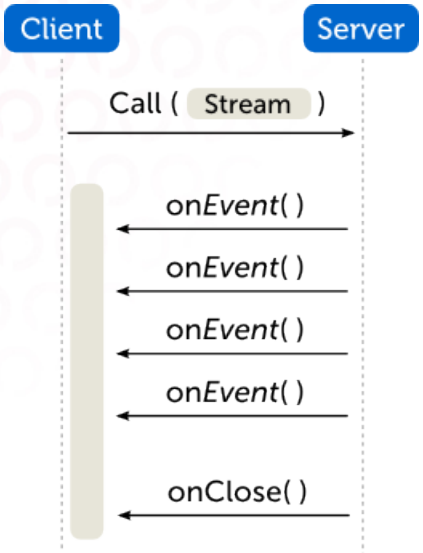


Figure 39: A stream

A stream is a response object that the client passes to an asynchronous call. When the call is ready to respond it calls a method on the stream to respond to the client. A stream differs from a callback, because it can receive more than one response. A stream continues to receive responses until it is closed or discarded.

All streams in the Unified API have the following methods:

on*Event*

Where *Event* depends on the specific stream and describes the event that occurs when the call is ready to return a response. This method is used to provide the responses from the call.

`onClose`

This method is used to indicate to the stream that all responses have been received from the call. After this method is called, the stream can not receive any more responses.

`onDiscard`

This method is used to indicate that the call has not returned a response, but that the callback is no longer required. This can happen when a call times out or a session is closed.

Contextual callbacks and streams

A contextual callback or contextual stream works in the same way as a callback or stream, except that when it is created an object is associated with it that provides application state information. For example, when a client requests information to populate a UI element it can associate an object that represents the UI element with the callback for the request. When the callback is called and provided with the information requested, it can use the associated context object to complete the appropriate action with that information: in this case, populating the UI element with the information.

Handlers

Handlers are part of the Diffusion interface that represent a server-side presence for a particular session.

Server handlers

These are components that a client application registers to handle events from the server that relate to the server and any other client sessions connected to the server.

The following uses are examples of the use of server handlers in the control client:

- To receive notifications of all clients that connect to or disconnect from the server.
- To authenticate other client sessions.
- To receive notifications regarding significant events on the queues of other clients.

A client is able to register only one server handler of any particular type.

All server handlers have a callback that is called on successful activation of the handler and is used to pass a registered handler reference, which the application can use to close the registration when required. There is also a callback that is used to notify that the registration has been closed.

Topic tree handlers

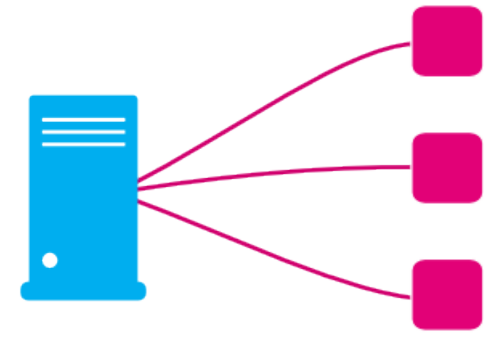
These are components that a control client registers to handle events from the server that relate to a particular part of the topic tree. Typically, when a handler is registered for a topic within the topic tree then the handler also applies to all topics beneath it within the topic tree unless superseded by another handler further down the tree.

The following uses are examples of the use of topic tree handlers in the control client:

- To handle some subscription requests from other clients
- To receive messages sent from other clients through topics
- To receive notifications of clients attempting to use topics that do not exist
- DEPRECATED: To handle requests from clients for the state of delegated topics.

All topic tree handlers have a callback that is called on successful activation of the handler and is used to pass a registered handler reference, which the application can use to close the registration when required. There is also an `onClose` callback, which is used to notify that the registration has been closed.

Load balancing



Multiple clients can connect to a single Diffusion server. If these clients have registered handlers on the same events at the server, the server can load balance requests between these clients.

Requests are load balanced between clients first by using a sticky-by-client approach – if a client has already handled a request from a standard client, further requests from the requesting client are routed to that client – and secondly using a round-robin approach – if no requests have previously been received from a client, the requesting client's first request is routed to the next handling client in the round robin.

Content

Content is the generic term for data that is transmitted between sessions and the Diffusion server. From the Diffusion point of view content is simply bytes and has no format imposed upon it by Diffusion.

Clients subscribe to topics receive the topic state and any updates as content. This content can be formatted according to the topic type. The API provides readers to simplify the interpretation of certain types of content.

Clients that send and receive messages through topics can also represent the data body part of the message as content, but can also add optional headers if required.

Clients can use content to initialize or update the state of topics at the server.

Even though content is simply treated as bytes there are aids to formatting content within the API. The content factory provides builders and convenience methods available for creating certain types of content, such as simple string content or Diffusion record-based content.

When a client uses content to update topics, it is important that the content is formatted in the way expected by the topic for updates.

Metadata in the Unified API

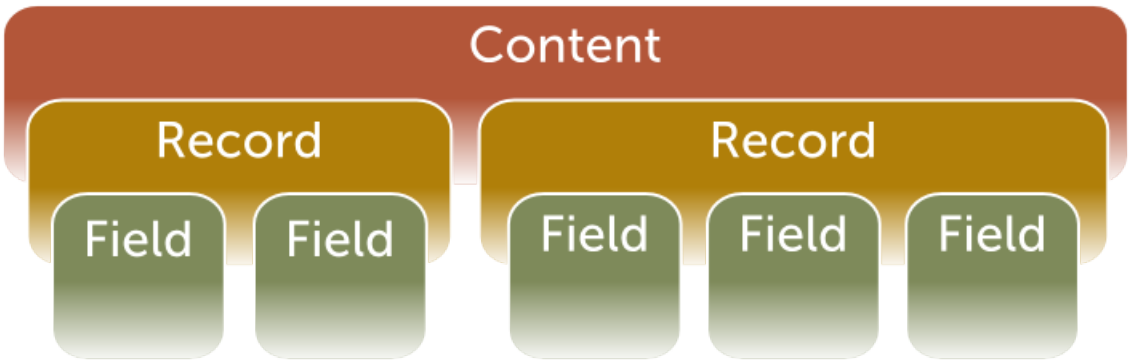
Metadata is available in the Unified API. It defines how the bytes in your content are formatted.

Content contains byte data. This byte data can be formatted in whatever way your application requires. For example, the data can structured using Diffusion record format.

You can use the provided Diffusion APIs to format the byte data as record-based data by defining a metadata structure that describes the data format.

Metadata structure

The metadata structure is made up of nested records and fields. The outer container is a the *content*. This contains zero, one, or many *records*. Each record can contain one or many *fields*.



Fields and records are identified by a name. Every record must have a name that is unique within the content. Every field must have a name that is unique within the enclosing record.

Every field or record defined in the metadata structure can represent one or more possible occurrences of that field or record in the byte data. The number of possible occurrences of a record or field is described by its multiplicity.

The order in which records and fields are defined within their enclosing container defines the order that they appear in byte data.

Field metadata

A metadata field defines an elementary data item within a record.

Every field has the following properties:

- Data type
- Multiplicity
- Default value

Data type

The data type of a field defines its actual representation within the byte data. The following table describes the data types that are available.

Table 48: Data types for metadata fields

Data type	Description	Default
String	A character string.	Zero-length string
Integer string	An integer represented in the content as a character string. If a field is defined as this type, it can only contain numeric digits with an optional leading sign. Fields of this type cannot be empty.	0
Decimal string	A decimal number represented in the content as a character string. Decimal fields have the number of places to the right of the decimal point defined by the scale, the default being 2. Such values can be parsed from a character string with any number of digits to the right of the decimal point. Half-up rounding is applied to achieve the target scale. Output of the field is rendered with the specified scale. Fields of this type cannot be empty. For comparison purposes the scale is ignored: a value of 1.50 is the same as 1.5.	0.00 (depending on scale)
Custom string	This is a special type where the behavior is delegated to a user-written custom field handler. This type is available in all topic data types.	-

Multiplicity

The multiplicity of a metadata field or record defines the number of times the corresponding byte data can occur within the enclosing record or content.

Multiplicity is defined in terms of the minimum and maximum number of occurrences. Some byte data representations support variable numbers of records and field, whereas others (such as record data) only support fixed number of records and fields (where minimum=maximum) except in the last position.

Fixed multiplicity is defined by a single number. For example, a multiplicity of 5 on a field indicates that there must be exactly five occurrences of the field within its enclosing record.

Variable multiplicity is defined by a minimum value and a maximum value and is represented with the notation *n*..*n*. For example, multiplicity of 1..5 on a field specifies that there can be between one and five occurrences of the field within its enclosing record.

A special maximum value of -1 is used to represent no maximum. For example, a multiplicity of 1..-1 on a field specifies there can be any number of occurrences of the field, but there must be at least one.

Optional nodes are defined by a minimum value of 0. For example, a multiplicity of 0..1 on a field specifies that there can be zero of one occurrences of the field within its enclosing record. A fixed multiplicity of 0 is not allowed.

Default value

You can specify a default value for a field. If you do not specify a default value, the default value for the data type is used. When content is created using metadata, default initialization applies the default values specified for each field.

Creating a metadata definition for a record topic
You can use the Unified API to specify the metadata structure that describes the byte data content of a message.

About this task

Publishing clients define the metadata structure for messages. This metadata structure can be used when defining a topic. All messages placed on the topic must conform to the metadata structure.

The Unified API for the following platforms provides builder methods that enable you to define the metadata structure:

- JavaScript
- Java
- .NET

The C Unified API does not provide builder methods, but instead takes the definition of the metadata as XML. For more information, see Using the C Unified API on page 471.

The following example demonstrates how to define the metadata structure using the Java Unified API.

> **Note:** Where there notation `c.p.d` is used in class or package names, it indicates `com.pushtechnology.diffusion`.

Procedure

1. Define the metadata structure.
   a) Import `c.p.d.client.Diffusion` and the following classes from the `c.p.d.client.content.metadata` package:
      - `MetadataFactory`
      - `MContent`
      - `MRecord`

- `MField`
- `MString`
- `MIntegerString`
- `MDecimalString`
- `MCustomString`

b) Use the `Diffusion.metadata` method to get a `MetadataFactory`.

```
private final MetadataFactory factory =
 Diffusion.metadata();
```

c) Use the methods on the `MetadataFactory` to specify the content, record, and field definitions that make up the metadata structure.

For example, the following code uses a content builder to create content metadata with a single record type that can occur zero to *n* times.

```
public MContent createContentRepeating() {
    return
        factory.contentBuilder("Content").
            add(
                factory.record(
                    "Rec1",
                    factory.string("A"),
                    factory.string("B")),
                0,
                -1).
            build();

}
```

For more information, see Java Unified API documentation.

2. Create a record topic and apply the metadata definition to it.

a) Import the TopicControl feature, `Session` class, and `RecordTopicDetails` class.

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicContr
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
```

b) Create a `Session` instance and use it to get the TopicControl feature.

```
final Session session = Diffusion.sessions().open("ws://
diffusion.example.com:80");
final TopicControl tc =
 session.feature(TopicControl.class);
```

c) Get a topic builder for a record topic from the TopicControl feature.

```
final RecordTopicDetails.Builder builder =

 tc.newDetailsBuilder(RecordTopicDetails.Builder.class);
```

d) Use the `metadata` method of the topic builder to create the topic definition.

```
tc.addTopic(
            TOPIC_NAME,
            builder.metadata(metadata).build(),
            new TopicControl.AddCallback.Default() {
                @Override
```

```
                public void onTopicAdded(String topic) {
                    theTopic = topic;
                }

            });
```

Example: A client that creates a metadata definition and uses it when creating a topic.

```
package com.example.metadata;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MContent;
import
 com.pushtechnology.diffusion.client.content.metadata.MDecimalString;
import
 com.pushtechnology.diffusion.client.content.metadata.MField;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
import
 com.pushtechnology.diffusion.client.types.UpdateOptions;


/**
 * An example of a client creating metadata definition and
 using it when creating a
 * topic definition.
 */
public class ControlClient {

    private final MetadataFactory factory =
 Diffusion.metadata();

    /**
     * Creates control client instance.
     */
    public ControlClient() {

        // Create the Session
        final Session session = Diffusion.sessions()
                .open("ws://diffusion.example.com:80");

        // Add the TopicControl feature
        final TopicControl tc =
 session.feature(TopicControl.class);

        // Create metadata for the topic
        final MContent metadata = defineMetadata();

        // Get a topic builder
        final RecordTopicDetails.Builder builder =

 tc.newDetailsBuilder(RecordTopicDetails.Builder.class);
```

```
        // Create the topic with metadata
        tc.addTopic(
            TOPIC_NAME,
            builder.metadata(metadata).build(),
            new TopicControl.AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    theTopic = topic;
                }

            });

    }


    /**
     * A simple example of creating content metadata with two
records.
     *
     * @return content metadata
     */
    public MContent defineMetadata() {
        return factory.content(
            "Content",
            createNameAndAddressRecord(),
            createMultipleRateCurrencyRecord("Exchange
Rates", 5));
    }

    /**
     * Creates a simple name and address record with fixed
name single
     * multiplicity fields.
     *
     * @return record definition.
     */
    public MRecord createNameAndAddressRecord() {
        return factory.record(
            "NameAndAddress",
            factory.string("FirstName"),
            factory.string("Surname"),
            factory.string("HouseNumber"),
            factory.string("Street"),
            factory.string("Town"),
            factory.string("State"),
            factory.string("PostCode"));
    }

    /**
     * This creates a record with two fields, a string called
"Currency" and a
     * decimal string called "Rate" with a default value of
1.00 which repeats a
     * specified number of times.
     *
     * @param name the record name
     * @param occurs the number of occurrences of the "Rate"
field
     * @return the metadata record
     */
    public MRecord createMultipleRateCurrencyRecord(String
name, int occurs) {
```

```
        return factory.recordBuilder(name).
            add(factory.string("Currency")).
            add(factory.decimal("Rate", "1.00"), occurs).
            build();
    }

}
```

Using a custom field handler

You can use the Java Classic API to implement a class that handles data in custom fields. This class is located on the server and can be invoked by the Unified API.

About this task

Note: Where there notation `c.p.d` is used in class or package names, it indicates `com.pushtechnology.diffusion`.

Custom field handlers implement the `c.p.d.api.data.metadata.CustomFieldHandler` interface.

Procedure

1. Create your custom field handler.

```
package com.example.custom

import
 com.pushtechnology.diffusion.api.data.metadata.CustomFieldHandler

public class DoubleFieldHandler implements CustomFieldHandler
 {

 public Object getInitialDefaultValue() {
  return new Double(0.0);
 }
 public Object parse(Object object) throws APIException {
  if (object==null) {
   return new Double(0.0);
  }

  try {
   return (Double.parseDouble(object.toString()));
  }
  catch (Throwable ex) {
   throw new APIException(
   "Unable to parse "+object+" as double value",
   ex);
  }
 }

 public boolean areEqual(Object source,Object target) {
  return source.equals(target);
 }
}
```

   a) Import `c.p.d.api.data.metadata.CustomFieldHandler`.
   b) Implement `CustomFieldHandler`.
   c) Implement the following methods:
      - `getInitialDefaultValue`, which returns the default value for the custom field

- `parse`, which parses any object into the data type used by the custom field
- `areEqual`, which compares two objects of the data type used by the custom field or equality

2. Put your compiled custom field handler class in *`diffusion_installation`*`/ext` directory and restart the Diffusion server.
3. Use the custom field handler when defining metadata for a custom string field from your control client.

```
package com.example.metadata;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MContent;
import
 com.pushtechnology.diffusion.client.content.metadata.MCustomString;
import
 com.pushtechnology.diffusion.client.content.metadata.MCustomString.Build
import
 com.pushtechnology.diffusion.client.content.metadata.MField;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;

public class ControlClient {

    private final MetadataFactory factory =
 Diffusion.metadata();

    /**
     * Create control client instance.
     */
    public ControlClient() {

    }

    /**
     * Example of using a custom field builder.
     * This method creates a custom string that uses the
 DoubleFieldHandler class
     * and contains the default value.
     */
    public MCustomString createCustom(String name) {
        return factory.customBuilder(name,
 "com.example.custom.DoubleFieldHandler").build();
    }

    /**
     * A simple example of creating content metadata with two
 records.
     * The records contain custom string fields.
     *
     * @return content metadata
     */
    public MContent createContent() {
        return factory.content(
            "Content",
            factory.record(
                "Rec1",
                createCustom("Field1"),
                createCustom("Field2")),
```

```
            factory.record(
                "Rec2",
                createCustom("Field3"));
    }

}
```

a) Import the following classes and interfaces:
   - `c.p.d.client.content.metadata.MetadataFactory;`
   - `c.p.d.client.content.metadata.MCustomString`
   - `c.p.d.client.content.metadata.MCustomString.Builder`

b) Get and instance of `MetadataFactory`.

c) Use the `customBuilder` method on the `MetadataFactory` to get a custom string builder. Pass in the name of the custom field and the fully qualified name of the custom field handler class you developed.

   For example:

```
factory.customBuilder(name,
 "com.example.custom.DoubleFieldHandler")
```

d) Use the `build` method on the `MCustomString.Builder` to get the `MCustomString`. You can use this metadata custom field definition when specifying your metadata structure.

Topic details

Topic details is the generic term used for a description of a topic.

There are the following different levels of topic detail:

BASIC

Provides the most basic detail about a topic, such as its type.

SCHEMA

Provides the basic detail plus information about the format of the data maintained by the topic. For topic types that have their content layout described by metadata, this is where the metadata is defined. Many topic types (such as functional topics) do not have schema definitions.

FULL

Provides the schema level of detail plus attributes of the topic. A client rarely requires this level of detail. It is necessary only when creating topics using a client.

When a client subscribes to any topic for the first time, it receives a notification of the basic topic details of that topic. This notification arrives before the first update is received for that topic.

Any client can also explicitly request any level of detail for any topic.

Clients use full topic details to describe a topic when creating it. Builders (and convenience methods) are available for creating details relating to all the different topic types.

When a client wants to create many different topics with the same description, we recommended that a single instance of a topic details object is used for every request. This is because there are caching optimizations at the server that mean that the same description is not transmitted to the server every time.

Topic types

The topic type defines what a topic is used for and how it behaves. Some topic types can have schema information and some can have attributes over and above the standard attributes available to all topic types.

The following topic types exist:

STATELESS

A topic that has no data held at the server and no particular function. It is generally used as an organizational node within the topic tree, but can also be used for sending messages.

> **Note:** When a topic is created with a hierarchic name, if the intermediate nodes in the hierarchy do not already exist they are automatically created as stateless topics.

DELEGATED

DEPRECATED: A topic that has no data held at the server. Its state is delegated to a control client. When a client subscribes or fetches, a control client that is registered to provide the state is invoked to retrieve the state. When a control client updates a delegated topic nothing is stored at the server and the update is simply fanned out to all subscribed clients.

SINGLE_VALUE

A topic that maintains state at the server as a single string value. The value can optionally be constrained in certain ways, for example, to hold an integer or decimal number. The nature of the single value is described using field metadata in the schema.

RECORD

A topic that maintains state at the server in Diffusion record format, which is effectively strings separated by field or record delimiters or both. This allows multiple fields to be maintained in the same topic and deltas of change are calculated at the server such that only those fields that have changed since the least update are sent out to the subscribed clients. The layout of the data is described using content metadata in the schema.

PROTOCOL_BUFFER

A topic that maintains state at the server in Google protocol buffers format. It benefits from the same delta processing capability at the server as a record topic. The schema defines a compiled proto class which must exist at the server and the name of a message definition within the class that defines the topic data layout. Google protocol buffer definitions cans be used to generate and interpret such content.

CUSTOM

A topic that has its state maintained server side by a user written Diffusion class. In many ways this makes it like a delegated topic except the state is delegated to an instance of the class and is all handled server side. Updates from a control client are also passed to the user handler for processing. The user handler can hold the topic state internally or elsewhere.

SLAVE

A special type of topic that has no state of its own but is a reference to the state of another topic. It is effectively a link to that primary topic. Updates to the primary topic are fanned out to subscribers of the slave. Updates cannot be made directly to slave topics. The primary topic must be one of the types of topic that maintains topic data.

SERVICE

A special type of topic that implements a request/response type service. The service is implemented as a user-written server-side Diffusion class.

PAGED_STRING

This is a special type of topic that maintains server-side state as a number of lines of string data. Clients using the Classic API can page through this data and have a current page that is updated if the state of that page changes. Paged topics can be created and updated using the Unified API.

PAGED_RECORD

A topic that is the same as a PAGED_STRING topic except each line of the data is defined in Diffusion record format. The schema defines the record metadata that defines the lines. Paged topics can be created and updated using the Unified API.

TOPIC_NOTIFY

A special type of topic that can be used by clients to receive notifications whenever topics are created or removed. Currently, though such a topic can be created using the Unified API, only clients implemented using the Classic API can use the topic.

ROUTING

A special type of topic, which can map to a different real topic for every client that subscribes to it. In this way, different clients can see different values for what is effectively the same topic from the client point of view. When a client subscribes to a routing topic, the request is either passed to a control client registered as a routing subscription handler for the topic or handled by a server-side routing handler.

CHILD_LIST

A special type of topic that maintains a list of its children and notifies subscribed clients when a child is added or removed. Though this type of topic can be created using the Unified API, it can be used only by Classic API clients.

Topic attributes

The attributes of a topic can be supplied when it is created, but in most cases the default values for the attributes are what is required.

There are a common set of attributes that apply to all topic types but certain types have additional attributes, some of which are mandatory. The API specifications of the relevant topic details builders indicate what is required.

The following topic attributes are common to all topic types:

Tidy on unsubscribe

By default, if a client unsubscribes from a topic, it continues to receive any updates to that topic that were queued for it before it unsubscribed. By choosing this option, when a client unsubscribes from the topic any updates queued for that topic are removed from the client's queue.

Reference

You can associate a string reference with a topic which is used in logging and monitoring to make the topic easier to identify.

Properties

DEPRECATED: An optional map of properties can be supplied in order to specify topic attributes not supported by the Unified API. This capability is present for backwards compatibility. The key values of the properties are the

same as the names in the `TopicProperty` enum in the Java Classic API and the values are string representations of the property value. This cannot be used to set properties that are represented in the Unified API.

Features

The Unified API is organized into features that make up conceptual groupings of capabilities.

These features that contain those capabilities that are primarily used by clients that connect to the Diffusion server to subscribe to topics and receive data:

Topics

- Subscribe to topics
- Fetch the state of topics

Messaging

- Send messages on topic paths
- Receive messages sent to the client session

Pings

- Ping the Diffusion server

Security

- Change the principal the client sesssion is authenticated with

Clients that use these features require roles with permission to access the topics they receive data through. For example, the CLIENT role.

Control features

Some features can be classified as control features.

These features contain those capabilities that are primarily used by clients that connect to the Diffusion server to update topics or administer the Diffusion server or other clients:

TopicControl

- Create topics
- Modify topics
- Delete topics

TopicUpdateControl

- Publish updates to topics (exclusively or non-exclusively)

MessagingControl

- Send messages to specific client sessions
- Handle messages sent via a specific topic path

AuthenticationControl

- Register handlers that authenticate other client sessions

SystemAuthenticationControl

- Update the system authentication store
- Define authentication decisions made by the Diffusion server
- Define the roles assigned to principals

SecurityControl

- Update the security store
- Define roles and their associated permissions

ClientControl

- Receive notifications about other client sessions
- Manage other client sessions

SubscriptionControl

- Subscribe other client sessions to topics

Clients that use control features require roles with additional permissions to use the capabilities of these features. For example, the ADMINISTRATOR role.

Topics

Use the Topics feature to enable a client session to subscribe to topics and receive streaming updates to those topics or to fetch the current state of topics.

The Topics feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
Topics	YES	YES	YES	YES

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Receiving data from topics

A client can use the Topics feature to subscribe to a topic or to fetch the state of a topic.

Subscribing to a topic

Required permissions: read_topic for the specified topic

A client can subscribe to a topic to receive updates that are published to the topic. If the topic has state, when the client subscribes to that topic it receives the topics state as a snapshot message. Subsequent updates to the data on the topic can be received as delta update messages or as snapshots depending on the type of the topic and the structure of its data.

Subscribing to multiple topics using a topic selector

Required permissions: read_topic for the specified topics

A client can subscribe to multiple topics in a single request by using topic selectors. Topic selectors enable you to select whole branches of the topic tree or use regular expressions to select topics based on the names in the topic path.

For more information, see Topic selectors in the Unified API on page 66.

Fetching the state of a topic

Required permissions: read_topic for the specified topic

A client can send a fetch request for the state of a topic. If the topic is of a type that maintains its state, the Diffusion server provides the current state of that topic to the client.

Example: Subscribe to a topic

The following examples use the Unified API to subscribe to topics and assign handlers to topics to receive the topic content.

JavaScript

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true
}).then(function(session) {

    // 1. Subscriptions are how sessions receive streams of data
 from the server.

    // When subscribing, a topic selector is used to select
 which topics to subscribe to. Topics do not need to exist
    // at the time of subscription - the server dynamically
 resolves subscriptions as topics are added or removed.

    // Subscribe to the "foo" topic with an inline callback
 function
    var subscription = session.subscribe('foo', function(update)
 {
        // Log the new value whenever the 'foo' topic is updated
        // By default, we get a Buffer object which preserves
 binary
        // data.
        console.log(update);
    });

    // Callbacks can also be registered after the subscription
 has occurred
    subscription.on({
        update : function(value, topic) {
            console.log('Update for topic: ' + topic, value);
        },
        subscribe : function(details, topic) {
            console.log('Subscribed to topic: ' + topic);
        },
        unsubscribe : function(reason, topic) {
            console.log('Unsubscribed from topic:' + topic);
        }
    });

    // 2. Sessions may unsubscribe from any topic to stop
 receiving data

    // Unsubscribe from the "foo" topic. Sessions do not need to
 have previously been subscribed to the topics they are
    // unsubscribing from. Unsubscribing from a topic will
 result in the 'unsubscribe' callback registered above being
    // called.
    session.unsubscribe('foo');

    // 3. Subscriptions / Unsubscriptions can select multiple
 topics using Topic Selectors

    // Topic Selectors provide regex-like capabilities for
 subscribing to topics. These are resolved dynamically, much
    // like subscribing to a single topic.
    var subscription2 = session.subscribe('?foo/.*/[a-z]');
```

```
    // 4. Subscriptions can use transformers to convert update
 values

    // Subscribe to a topic and then convert all received values
 to JSON. Transforming a subscription creates a new
    // subscription stream, rather than modifying the original.
    session.subscribe('bar').transform(JSON.parse).on('update',
 function(value, topic) {
        console.log('Got JSON update for topic: ' + topic,
 value);
    });

    // 5. Metadata can be used within transformers to parse data

    // Create a simple metadata instance
    var meta = new diffusion.metadata.RecordContent();

    // Add a single record/field
    meta.addRecord('record', {
        'field' : meta.string('some-value')
    });

    // Subscribe to a topic and transform with the metadata
    session.subscribe('baz').transform(meta).on('update',
 function(value) {
        console.log('Field value: ',
 value.get('record').get('field'));
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.content.RecordContentReader;
import com.pushtechnology.diffusion.client.features.Topics;
import
 com.pushtechnology.diffusion.client.features.Topics.TopicStream;
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.types.UpdateContext;

/**
 * In this simple and commonest case for a client we just
 subscribe to a few
 * topics and assign handlers for each to receive content.
 * <P>
 * This makes use of the 'Topics' feature only.
 * <P>
 * To subscribe to a topic, the client session must have the
 'read_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
```

```
public final class ClientSimpleSubscriber {

    private static final Logger LOG =
        LoggerFactory.getLogger(ClientSimpleSubscriber.class);

    private final Session session;

    /**
     * Constructor.
     */
    public ClientSimpleSubscriber() {

        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");

        // Use the Topics feature to add a topic stream for
        // Foo and all topics under Bar and request subscription
 to those topics
        final Topics topics = session.feature(Topics.class);
        topics.addTopicStream(">Foo", new FooTopicStream());
        topics.addTopicStream(">Bar/", new BarTopicStream());
        topics.subscribe(
            Diffusion.topicSelectors().anyOf("Foo", "Bar//"),
            new Topics.CompletionCallback.Default());
    }

    /**
     * Close session.
     */
    public void close() {
        session.close();
    }

    /**
     * The topic stream for all messages on the 'Foo' topic.
     */
    private class FooTopicStream extends TopicStream.Default {
        @Override
        public void onTopicUpdate(
            String topic,
            Content content,
            UpdateContext context) {

            LOG.info(content.asString());
        }
    }

    /**
     * The topic stream for all messages on 'Bar' topics.
     */
    private class BarTopicStream extends TopicStream.Default {
        @Override
        public void onTopicUpdate(
            String topic,
            Content content,
            UpdateContext context) {

            // Process the message - one with a record with a
 variable number of
            // fields followed by two more fields (effectively
 another record
```

```
            // but no need to process as such).

            final RecordContentReader reader =
                Diffusion.content().newReader(
                    RecordContentReader.class,
                    content);

            for (String field : reader.nextRecord()) {
                LOG.info("Record 1 Field={}", field);
            }

            LOG.info("Extra Field 1={}", reader.nextField());
            LOG.info("Extra Field 2={}", reader.nextField());
        }
    }
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Content.Metadata;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
    /// <summary>
    /// In this simple and commonest case for a client, we just
 subscribe to a few topics and assign handlers for each
    /// to receive content.
    ///
    /// This makes use of the <see cref="ITopics"/> feature
 only.
    ///
    /// To subscribe to a topic, the client session must have
 the <see cref="TopicPermission.READ_TOPIC"/> permission
    /// for that branch of the topic tree.
    /// </summary>
    public class ClientSimpleSubscriber
    {
        #region Fields

        private readonly ISession session;

        #endregion Fields

        #region Constructor

        /// <summary>
        /// Constructor.
        /// </summary>
        public ClientSimpleSubscriber()
        {
            session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );
```

```
            // Use the Topics feature to add a topic stream for
 Foo and all topics under Bar and request subscription
            // to those topics
            var topics = session.GetTopicsFeature();

            topics.AddTopicStream( ">Foo", new
 FooTopicStream() );
            topics.AddTopicStream( ">Bar/", new
 BarTopicStream() );


 topics.Subscribe( Diffusion.TopicSelectors.AnyOf( "Foo",
 "Bar//" ), new TopicsCompletionCallbackDefault() );
        }

        #endregion Constructor

        #region Public Methods

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close()
        {
            session.Close();
        }

        #endregion Public Methods

        #region Private Classes

        /// <summary>
        /// The topic stream for all messages on the 'Foo'
 topic.
        /// </summary>
        private class FooTopicStream : TopicStreamDefault
        {
            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public override void OnTopicUpdate( string
 topicPath, IContent content, IUpdateContext context )
            {
            }
        }

        /// <summary>
        /// The topic stream for all messages on 'Bar' topics.
        /// </summary>
```

```
        private class BarTopicStream : TopicStreamDefault
        {
            #region Fields

            #endregion Fields

            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public override void OnTopicUpdate( string
 topicPath, IContent content, IUpdateContext context )
            {
                var reader =
 Diffusion.Content.NewReader<IRecordContentReader>( content );

                foreach( var field in reader.NextRecord() )
                {
                    // Log the value of each field here.
                }
            }
        }

        private class TopicStreamDefault : ITopicStream
        {
            /// <summary>
            /// Notification of a contextual error related to
 this callback. This is
            /// analogous to an exception being raised.
 Situations in which
            /// <code>OnError</code> is called include the
 session being closed, a
            /// communication timeout, or a problem with the
 provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public virtual void OnError( ErrorReason
 errorReason )
            {
            }

            /// <summary>
            /// Called to notify that a stream context was
 closed normally.
            ///
```

```
            /// No further calls will be made for the stream
 context.
            /// </summary>
            public void OnClose()
            {
            }

            /// <summary>
            /// This notifies when a topic is subscribed to.
            ///
            /// This provides only <see
 cref="TopicDetailsLevel.BASIC"/>details of the topic.
            /// </summary>
            /// <param name="topicPath">the full topic path</
param>
            /// <param name="details">the basic details</param>
            public virtual void OnSubscription( string
 topicPath, ITopicDetails details )
            {
            }

            /// <summary>
            /// This notifies when a topic is unsubscribed.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="reason">the reason for
 unsubscription.</param>
            public virtual void OnUnsubscription( string
 topicPath, TopicUnsubscribeReason reason )
            {
            }

            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public virtual void OnTopicUpdate( string topicPath,
 IContent content, IUpdateContext context )
            {
            }
        }

        #endregion Private Classes
    }
}
```

C

```
/*
 * This is a sample client which connects to Diffusion v5 and
 subscribes to
 * topics using a user-specified selector. Any messages received
 on those topics
 * are then displayed to standard output.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/*
 * This callback is used when the session state changes, e.g.
 when a session
 * moves from a "connecting" to a "connected" state, or from
 "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session,
                         const SESSION_STATE_T old_state,
                         const SESSION_STATE_T new_state)
{
        printf("Session state changed from %s (%d) to %s
 (%d)\n",
               session_state_as_string(old_state), old_state,
               session_state_as_string(new_state), new_state);
}

/*
 * When a subscribed message is received, this callback is
 invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
        printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
        return HANDLER_SUCCESS;
}

/*
 * This callback is fired when Diffusion responds to say that a
 topic
 * subscription request has been received and processed.
 */
static int
on_subscribe(SESSION_T *session, void *context_data)
```

```
{
        printf("on_subscribe\n");
        return HANDLER_SUCCESS;
}

/*
 * This is callback is for when Diffusion response to an
 unsubscription
 * request to a topic, and only indicates that the request has
 been received.
 */
static int
on_unsubscribe(SESSION_T *session, void *context_data)
{
        printf("on_unsubscribe\n");
        return HANDLER_SUCCESS;
}

/*
 * Publishers and control clients may choose to subscribe any
 other client to
 * a topic of their choice at any time. We register this
 callback to capture
 * messages from these topics and display them.
 */
static int
on_unexpected_topic_message(SESSION_T *session, const
 TOPIC_MESSAGE_T *msg)
{
        printf("Received a message for a topic we didn't
 subscribe to (%s)\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
        return HANDLER_SUCCESS;
}

/*
 * We use this callback when Diffusion notifies us that we've
 been subscribed
 * to a topic. Note that this could be called for topics that we
 haven't
 * explicitly subscribed to - other control clients or
 publishers may ask to
 * subscribe us to a topic.
 */
static int
on_notify_subscription(SESSION_T *session, const
 SVC_NOTIFY_SUBSCRIPTION_REQUEST_T *request, void *context)
{
        printf("on_notify_subscription: %d: \"%s\"\n",
               request->topic_info.topic_id,
               request->topic_info.topic_path);
        return HANDLER_SUCCESS;
}

/*
 * This callback is used when we receive notification that this
 client has been
 * unsubscribed from a specific topic. Causes of the
 unsubscription are the same
 * as those for subscription.
 */
static int
```

```
on_notify_unsubscription(SESSION_T *session, const
 SVC_NOTIFY_UNSUBSCRIPTION_REQUEST_T *request, void *context)
{
        printf("on_notify_unsubscription: %d, reason: %d\n",
               request->topic_id,
               request->reason);
        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
        // Standard command line parsing
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        char *topic = hash_get(options, "topic_selector");

        // A SESSION_LISTENER_T holds callbacks to inform the
 client
        // about changes to the state. Used here for
 informational
        // purposes only.
        SESSION_LISTENER_T session_listener;
        session_listener.on_state_changed =
 &on_session_state_changed;

        // Creating a session requires at least a URL. Creating
 a session
        // initiates a connection with Diffusion.
        DIFFUSION_ERROR_T error;
        SESSION_T *session = NULL;
        session = session_create(url, NULL, NULL,
 &session_listener, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        // When issuing commands to Diffusion (in this case,
 subscribe to
        // a topic), it's typical that more than one message may
 be
        // received in response and a handler can be installed
 for each
        // message type. In the case of subscription, we can
 install
        // handlers for:
        // 1. The topic message data (on_topic_message).
        // 2. Notification that the subscription has been
 received (on_subscribe).
        // 3. Topic details (on_topic_details).

        notify_subscription_register(session,
 (NOTIFY_SUBSCRIPTION_PARAMS_T) { .on_notify_subscription =
 on_notify_subscription });
```

```
        notify_unsubscription_register(session,
 (NOTIFY_UNSUBSCRIPTION_PARAMS_T) { .on_notify_unsubscription =
 on_notify_unsubscription });

        subscribe(session, (SUBSCRIPTION_PARAMS_T)
 { .topic_selector = topic, .on_topic_message =
 on_topic_message, .on_subscribe = on_subscribe });

        // Install a global topic handler to capture messages
 for topics we haven't
        // explicitly subscribed to, and therefore don't have a
 specific handler for.
        session->global_topic_handler =
 on_unexpected_topic_message;

        // Keep receiving messages for 5 seconds.
        sleep(5);

        // Unsubscribe from the topic
        unsubscribe(session, (UNSUBSCRIPTION_PARAMS_T)
 {.topic_selector = topic, .on_unsubscribe = on_unsubscribe} );

        // Wait for any unsubscription notifications to be
 received.
        sleep(5);

        // Politely tell Diffusion we're closing down.
        session_close(session, &error);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related Links

Topic selectors in the Unified API on page 66
A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topic selectors in the Classic API on page 72
A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Example: Fetch topic state

The following examples use the Unified API to fetch the current state of a topic without subscribing to the topic.

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Topics;
import
 com.pushtechnology.diffusion.client.features.Topics.FetchContextStream;
import com.pushtechnology.diffusion.client.session.Session;

/**
```

```
 * This is a simple example of a client that fetches the state
 of topics but
 * does not subscribe to them.
 * <P>
 * This makes use of the 'Topics' feature only.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientUsingFetch {

    private final Session session;
    private final Topics topics;

    /**
     * Constructor.
     */
    public ClientUsingFetch() {

        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");

        topics = session.feature(Topics.class);
    }

    /**
     * Issues a fetch request for a topic or selection of
 topics.
     *
     * @param topicSelector a {@link TopicSelector} expression
     * @param fetchContext context string to be returned with
 the fetch
     *        response(s)
     * @param stream callback for fetch responses
     */
    public void fetch(
        String topicSelector,
        String fetchContext,
        FetchContextStream<String> stream) {

        topics.fetch(topicSelector, fetchContext, stream);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.DiffusionCore.Messaging.Topic;

namespace UCIStack.Examples
```

```
{
 /// <summary>
 /// This is a simple example of a client that fetches the state
 of topics but does not subscribe to them.
 ///
 /// This makes use of the <see cref="ITopics"/> feature only.
 /// </summary>
 public class ClientUsingFetch
 {
  #region Fields

  private readonly ISession session;
  private readonly ITopics topics;

  #endregion Fields

  #region Constructor

  public ClientUsingFetch()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topics = session.GetTopicsFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Issues a fetch request for a topic or selection of topics.
  /// </summary>
  /// <param name="topicSelector">A <see cref="TopicSelector"/>
 expression.</param>
  /// <param name="fetchContext">The context string to be
 returned with the fetch response(s).</param>
  /// <param name="stream">The callback for fetch responses.</
param>
  public void Fetch( string topicSelector, string fetchContext,
 IFetchContextStream<string> stream )
  {
   topics.Fetch( topicSelector, fetchContext, stream );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

C

```
/*
```

```
 * This is a sample client which connects to Diffusion v5 and
 demonstrates
 * the following features:
 *
 * 1. Fetch topic state using a user-specified topic selector.
 * 2. Connect to Diffusion with a username and password.
 * 3. Automatic retry of a connection if unable to connect at
 the first
 *    attempt.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

extern void topic_message_debug();

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        {'r', "retries", "Number of connection retries",
 ARG_OPTIONAL, ARG_HAS_VALUE, "3"},
        {'d', "retry_delay", "Delay (in ms) between connection
 attempts", ARG_OPTIONAL, ARG_HAS_VALUE, "1000"},
 {'p', "principal", "Principal (username) for the connection",
 ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
 {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
 * This callback is used when the session state changes, e.g.
 when a session
 * moves from a "connecting" to a "connected" state, or from
 "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session, const
 SESSION_STATE_T old_state, const SESSION_STATE_T new_state)
{
        printf("Session state changed from %s (%d) to %s
 (%d)\n",
                session_state_as_string(old_state), old_state,
                session_state_as_string(new_state), new_state);
        if(new_state == CONNECTED_ACTIVE) {
                printf("Session ID=%s\n",
 session_id_to_string(session->id));
        }
}

/**
 * This callback is invoked when Diffusion acknowledges that it
 has received
 * the fetch request. It does not indicate that there will be
 any subsequent
```

```
 * messages; see on_topic_message() and
 on_fetch_status_message() for that.
 */
static int
on_fetch(SESSION_T *session, void *context)
{
 puts("Fetch acknowledged by server");
 return HANDLER_SUCCESS;
}

/**
 * This callback is invoked when all messages for a topic
 selector have
 * been received, or there was some kind of server-side error
 during the
 * fetch processing.
 */
static int
on_fetch_status_message(SESSION_T *session,
                        const SVC_FETCH_STATUS_RESPONSE_T
 *status,
                        void *context)
{
 switch(status->status_flag) {
 case DIFFUSION_TRUE:
  puts("Fetch succeeded");
  break; //exit(0);
 case DIFFUSION_FALSE:
  puts("Fetch failed");
  break; //exit(1);
 default:
  printf("Unknown fetch status: %d\n", status->status_flag);
  break;
 }

 return HANDLER_SUCCESS;
}

/**
 * When a fetched message is received, this callback in invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
 printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);

#ifdef DEBUG
         topic_message_debug(response->payload);
#endif

 return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
```

```
                return 1;
        }

        char *url = hash_get(options, "url");
        char *topic = hash_get(options, "topic_selector");
        int retries = atoi(hash_get(options, "retries"));
        long retry_delay = atol(hash_get(options,
"retry_delay"));

// A SESSION_LISTENER_T holds callbacks to inform the client
        // about changes to the state. Used here for
informational
        // purposes only.
        SESSION_LISTENER_T foo_listener;
        foo_listener.on_state_changed =
&on_session_state_changed;

// The client-side API can automatically keep retrying to
connect
// to the Diffusion server if it's not immediately available.
        SESSION_FAILOVER_STRATEGY_T failover_strategy;
        failover_strategy.retry_count = retries;
        failover_strategy.retry_delay = retry_delay;

// Creating a session requires at least a URL. Creating a
session
        // initiates a connection with Diffusion.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url,
                                 hash_get(options, "principal"),

credentials_create_password(hash_get(options, "credentials")),
                                 &foo_listener,
&failover_strategy, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

// Register handlers for callbacks we're interested in relating
to
// the fetch request. In particular, we want to know about the
topic
// messages that are returned, and the status message which
tells
// us when all messages have been received for the selector
(or, if
// something went wrong.)
        FETCH_PARAMS_T params = {
                .selector = topic,
                .on_topic_message = on_topic_message,
                .on_fetch = on_fetch,
                .on_status_message = on_fetch_status_message
        };

// Issue the fetch request.
fetch(session, params);

// Wait for up to 5 seconds for the results to come in.
        sleep(1);
```

```
 // Clean up politely.
        session_close(session, &error);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Messaging

Use the Messaging feature to enable a client session to send messages to a topic path.

The Messaging feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
Messaging	YES	YES	YES	YES

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Messaging to topic paths

A client can use the Messaging feature to send individual messages to a handler that has registered to receive messages on a topic path.

Sending messages to clients

Required permissions: send_to_message_handler

A client can send a message to a topic path, regardless of whether a topic is bound to that topic path. The messages are delivered to a handler that has registered to receive messages on that topic path.

The body of the message that is sent is represented as *content*. Any of the content builder features can be used to build message content. With messaging you can also send empty content.

When sending a message certain additional options can also be specified:

Headers

A set of string values that can be sent along with the content.

Priority

Use this to specify the priority used when queuing the message for the client at the the Diffusion server.

Listen for messages on a topic path

A client can specify a listener that receives messages sent to the client on a topic path.

Example: Send a message to a topic path

The following examples use the Unified API to send a message to a topic path. The message is received by a handler that has registered to receive messages on that topic path.

JavaScript

```
var diffusion = require('diffusion');
```

```
// Connect to the server. Change these options to suit your own
 environment.
// Node.js will not accept self-signed certificates by default.
 If you have
// one of these, set the environment variable
 NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. Messages can be sent & received between sessions.

    // Create a stream of received messages for a specific path
    session.messages.listen('foo').on('message', function(msg) {
        console.log('Received message: ' + msg.content);
    });

    // Send a message to another session. It is the
 application's responsibility to find the SessionID of the
 intended
    // recipient.
    session.messages.send('foo', 'Hello world', 'another-
session-id');

    // 2. Messages can also be sent without a recipient, in
 which case they will be dispatched to any Message Handlers
    // that have been registered for the same path. If multiple
 handlers are registered to the same path, any given
    // message will only be dispatched to one handler.

    // Register the handler to receive messages at or below the
 given path.
    session.messages.addHandler('foo', {
        onActive : function() {
            console.log('Handler registered');
        },
        onClose : function() {
            console.log('Handler closed');
        },
        onMessage : function(msg) {
            console.log('Received message:' + msg.content + '
 from Session: ' + msg.session);
            if (msg.properties) {
                console.log('with properties:', msg.properties);
            }
        }
    }).then(function() {
        console.log('Registered handler');
    }, function(e) {
        console.log('Failed to register handler: ', e);
    });

    // Send a message at a lower path, without an explicit
 recipient - this will be received by the Handler.
    session.messages.send('foo/bar', 'Another message');
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.List;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Messaging;
import
 com.pushtechnology.diffusion.client.features.Messaging.SendCallback;
import
 com.pushtechnology.diffusion.client.features.Messaging.SendContextCallback
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This is a simple example of a client that uses the
 'Messaging' feature to
 * send messages to a topic path.
 * <P>
 * To send a message on a topic path, the client session
 requires the
 * 'send_to_message_handler' permission.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientSendingMessages {

    private final Session session;
    private final Messaging messaging;

    /**
     * Constructs a message sending application.
     */
    public ClientSendingMessages() {
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        messaging = session.feature(Messaging.class);
    }

    /**
     * Sends a simple string message to a specified topic path.
     * <P>
     * There will be no context with the message so callback
 will be directed to
     * the no context callback.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param callback notifies message sent
     */
    public void send(String topicPath, String message,
 SendCallback callback) {
        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),
            callback);
    }

    /**
```

```
     * Sends a simple string message to a specified topic path
with context string.
     * <P>
     * Callback will be directed to the contextual callback with
the string
     * provided.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param context the context string to return with the
callback
     * @param callback notifies message sent
     */
    public void send(
        String topicPath,
        String message,
        String context,
        SendContextCallback<String> callback) {

        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),
            context,
            callback);
    }

    /**
     * Sends a string message to a specified topic path with
headers.
     * <P>
     * There will be no context with the message so callback
will be directed to
     * the no context callback.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param headers the headers to send with the message
     * @param callback notifies message sent
     */
    public void sendWithHeaders(
        String topicPath,
        String message,
        List<String> headers,
        SendCallback callback) {

        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),

messaging.sendOptionsBuilder().headers(headers).build(),
            callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

.NET

```
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a simple example of a client that uses the
 'Messaging' feature to send messages on a topic path.
 ///
 /// To send messages on a topic path,
 the client session requires the <see
 cref="TopicPermission.SEND_TO_MESSAGE_HANDLER"/>
 /// permission.
 /// </summary>
 public class ClientSendingMessages
 {
  #region Fields

  private readonly ISession session;
  private readonly IMessaging messaging;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructs a message sending application.
  /// </summary>
  public ClientSendingMessages()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   messaging = session.GetMessagingFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Sends a simple string message to a specified topic path.
  ///
  /// There will be no context with the message so callback will
 be directed to the 'no context' callback.
  /// </summary>
  /// <param name="topicPath">The topic path.</param>
  /// <param name="message">The message to send.</param>
  /// <param name="callback">Notifies that the message was
 sent.</param>
  public void Send( string topicPath, string message,
 ISendCallback callback )
  {
   messaging.Send( topicPath,
 Diffusion.Content.NewContent( message ), callback );
  }
```

```
  /// <summary>
  /// Sends a simple string message to a specified topic path
 with context string.
  ///
  /// The callback will be directed to the contextual callback
 with the string provided.
  /// </summary>
  /// <param name="topicPath"></param>
  /// <param name="message"></param>
  /// <param name="context"></param>
  /// <param name="callback"></param>
  public void Send( string topicPath, string message, string
 context, ISendContextCallback<string> callback )
  {
   messaging.Send( topicPath,
 Diffusion.Content.NewContent( message ), context, callback );
  }

  /// <summary>
  /// Sends a string message to a specified topic with headers.
  ///
  /// There will be no context with the message so callback will
 be directed to the 'no context' callback.
  /// </summary>
  /// <param name="topicPath">The topic path.</param>
  /// <param name="message">The message to send.</param>
  /// <param name="headers">The headers to send with the
 message.</param>
  /// <param name="callback">Notifies that the message was
 sent.</param>
  public void SendWithHeaders( string topicPath, string message,
 List<string> headers, ISendCallback callback )
  {
   messaging.Send(
    topicPath,
    Diffusion.Content.NewContent( message ),

 messaging.CreateSendOptionsBuilder().SetHeaders( headers ).Build(),
    callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

C

```
#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
```

```
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'t', "topic", "Topic name", ARG_REQUIRED,
 ARG_HAS_VALUE, "echo"},
        {'d', "data", "Data to send", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
 * Callback invoked when/if a message is published on the topic
 that the
 * client is writing to.
 */
static int
topic_handler(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
        printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
 return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }
        char *topic = hash_get(options, "topic");

 // For subscription, we create a selector which specifies the
 topic
 // we're sending to *only*.
        char *topic_selector = malloc(1 + strlen(topic) + 1);
        topic_selector[0] = '>';
        strcpy(&topic_selector[1], topic);

 // Create a session with Diffusion.
        SESSION_T *session = NULL;
        DIFFUSION_ERROR_T error;
        session = session_create(url, principal, credentials,
 NULL, NULL, &error);
        if(session == NULL) {
```

```
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

 // Install a subscription handler.
 SUBSCRIPTION_HANDLERS_T *handlers = calloc(1,
 sizeof(SUBSCRIPTION_HANDLERS_T));
 handlers->on_topic_message = &topic_handler;
 subscribe(session, (SUBSCRIPTION_PARAMS_T) { .topic_selector =
 topic_selector, .on_topic_message = topic_handler });

 // Create a message.
        char *data = hash_get(options, "data");

        LIST_T *headers = list_create();
        BUF_T *payload = buf_create();
        buf_write_bytes(payload, data, strlen(data));

        MESSAGE_T msg = {
                .type = MESSAGE_TYPE_TOPIC_LOAD,
                .headers = headers,
                .payload = payload
        };

 // Send the message.
        SEND_MSG_PARAMS_T params = {
                .topic_path = hash_get(options, "topic"),
                .msg = msg,
                .priority = MESSAGE_PRIORITY_MEDIUM
        };
        send_msg(session, params);

 // Wait a few seconds; send_msg is asynchronous and it we don't
 want
 // to kill the session until the message has been given chance
 to be
 // sent. We also want to wait for any possible responses.
        sleep(5);

 // Politely close the client connection.
        session_close(session, &error);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Pings

Use the Pings feature to enable a client session to ping the server and verify the connection between client and server.

The Pings feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
Pings	NO (Handled automatically)	YES	YES	YES

The following examples use the Unified API to ping the server from a client:

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Pings;
import
 com.pushtechnology.diffusion.client.features.Pings.PingContextCallback;
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This is a simple client example that pings the server and
 prints out the
 * round-trip time.
 * <P>
 * This uses the 'Pings' feature only.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientUsingPings {

    private final Session session;
    private final Pings pings;

    /**
     * Constructor.
     */
    public ClientUsingPings() {
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        pings = session.feature(Pings.class);
    }

    /**
     * Ping the server.
     *
     * @param context string to log with round trip time
     * @param callback used to return ping reply
     */
    public void ping(String context, PingContextCallback<String>
 callback) {
        pings.pingServer(context, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
```

```
namespace UCIStack.Examples
{
 /// <summary>
 /// This is a simple client example that pings the server and
 prints out the round-trip time.
 ///
 /// This uses the <see cref="IPings"/> feature only.
 /// </summary>
 public class ClientUsingPings
 {
  #region Fields

  private readonly ISession session;
  private readonly IPings pings;

  #endregion Fields

  #region Constructor

  public ClientUsingPings()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   pings = session.GetPingFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Ping the server.
  /// </summary>
  /// <param name="context">The string to log with round-trip
 time.</param>
  /// <param name="callback">Used to return the ping reply.</
param>
  public void Ping( string context, IPingContextCallback<string>
 callback )
  {
   pings.PingServer( context, callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

C

```
#include <stdio.h>
#include <unistd.h>
```

```
#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        END_OF_ARG_OPTS
};

/**
 * Callback for displaying the receipts of a ping response.
 */
static int
on_ping_response(SESSION_T *session, const SVC_PING_RESPONSE_T
 *response, void *context)
{
        printf("Received ping response\n");
        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
        // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");

        // Create a session with the Diffusion server.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, NULL, NULL, NULL, NULL,
 &error);
        if(session == NULL) {
                fprintf(stderr, "Failed to create session: %s
\n", error.message);
                return 1;
        }

        // Define a ping response handler
        PING_PARAMS_T ping_params = {
                .on_ping_response = on_ping_response
        };

        // Send 5 pings at 1 sec intervals
        int i;
        for(i = 0; i < 5; i++) {
                ping(session, ping_params);
                sleep(1);
        }

        // Gracefully close the client session.
        session_close(session, &error);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Security

Use the Security feature to enable a client to change the principal associated with its session.

The Security feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
Security	YES	YES	YES	YES

The following examples use the Unified API to change the client principal and credentials for a client session:

JavaScript

Note: Only step 1 deals with changing the principal.

```
// Session security allows you to change the principal that a
 session is authenticated as. It also  allows users to
// query and update server-side security and authentication
 stores, which control users, roles and permissions.
// This enables you to manage the capabilities that any logged
 in user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {

    // 1. A session change their principal by re-authenticating
    session.security.changePrincipal('admin',
 'password').then(function() {
        console.log('Authenticated as admin');
    });

    // 2. The security configuration provides details about
 roles and their assigned permissions

 session.security.getSecurityConfiguration().then(function(config)
 {
        console.log('Roles for anonymous sessions: ',
 config.anonymous);
        console.log('Roles for named sessions: ', config.named);
        console.log('Available roles: ', config.roles);
    }, function(error) {
        console.log('Unable to fetch security configuration',
 error);
    });
```

```
    // 3. Changes to the security configuration are done with a
SecurityScriptBuilder
    var securityScriptBuilder =
session.security.securityScriptBuilder();

    // Set the permissions for a particular role - global and
topic-scoped
    // Each method on a script builder returns a new builder
    var setPermissionScript =
securityScriptBuilder.setGlobalPermissions('SUPERUSER',
['REGISTER_HANDLER'])

.setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
                                                    .build();

    // Update the server-side store with the generated script

session.security.updateSecurityStore(setPermissionScript).then(function()
{
        console.log('Security configuration updated
successfully');
    }, function(error) {
        console.log('Failed to update security configuration: ',
error);
    });

    // 4. The system authentication configuration lists all
users & roles

session.security.getSystemAuthenticationConfiguration().then(function(conf
{
        console.log('System principals: ', config.principals);
        console.log('Anonymous sessions: ', config.anonymous);
    }, function(error) {
        console.log('Unable to fetch system authentication
configuration', error);
    });

    // 5. Changes to the system authentication config are done
with a SystemAuthenticationScriptBuilder
    var authenticationScriptBuilder =
session.security.authenticationScriptBuilder();

    // Add a new user and set password & roles.
    var addUserScript =
authenticationScriptBuilder.addPrincipal('Superman',
'correcthorsebatterystapler')

.assignRoles('Superman', ['SUPERUSER'])
                                                    .build();

    // Update the system authentication store
    session.security.updateStore(addUserScript).then(function()
{
        console.log('Updated system authentication config');
    }, function(error) {
        console.log('Failed to update system authentication: ',
error);
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Security;
import
 com.pushtechnology.diffusion.client.features.Security.ChangePrincipalCallb
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This demonstrates a client's use of credentials, specifically
 the ability to
 * change the principal for an active session.
 * <P>
 * This is not a realistic use case on its own, but is shown
 separately here for
 * clarity.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ClientUsingCredentials {

    private final Session session;
    private final Security security;

    /**
     * Constructor.
     */
    public ClientUsingCredentials() {
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        security = session.feature(Security.class);
    }

    /**
     * Request a change of principal for the session.
     *
     * @param principal the new principal name
     * @param password the password
     * @param callback notifies success or failure
     */
    public void changePrincipal(
        String principal,
        String password,
        ChangePrincipalCallback callback) {

        security.changePrincipal(
            principal,
            Diffusion.credentials().password(password),
            callback);
    }

    /**
     * Close.
     */
    public void close() {
        session.close();
    }
```

```
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates a client's use of credentials,
 specifically the ability to change the principal for an active
 /// session.
 ///
 /// This is not a realistic use case on its own, but it shown
 separately here for clarity.
 /// </summary>
 public class ClientUsingCredentials
 {
  #region Fields

  private readonly ISession session;
  private readonly ISecurity security;

  #endregion Fields

  #region Constructor

  public ClientUsingCredentials()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   security = session.GetSecurityFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Request a change of principal for the session.
  /// </summary>
  /// <param name="principal">The new principal name.</param>
  /// <param name="password">The password.</param>
  /// <param name="callback">Notifies success or failure.</
param>
  public void ChangePrincipal( string principal, string
 password, IChangePrincipalCallback callback )
  {
   security.ChangePrincipal( principal,
 Diffusion.Credentials.Password( password ), callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
```

```
  }

  #endregion Public Methods
 }
}
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/*
 * Callback to display that the change_principal() request has
 been processed
 * by Diffusion.
 */
static int
on_change_principal(SESSION_T *session, void *context)
{
        printf("on_change_principal\n");
        return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
        // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");

        // Create a session with Diffusion, with no principal or
 credentials.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, NULL, NULL, NULL, NULL,
 &error);
        if(session == NULL) {
                fprintf(stderr, "Failed to create session: %s
\n", error.message);
                return 1;
        }
```

```
        // Wait for a couple of seconds.
        sleep(2);

        puts("Changing credentials");

        CREDENTIALS_T *credentials =
 credentials_create_password("chips");

        // Specify callbacks for the change_principal request.
        CHANGE_PRINCIPAL_PARAMS_T params = {
                .principal = "fish",
                .credentials = credentials,
                .on_change_principal = on_change_principal
        };

        // Do the change.
        change_principal(session, params);

        // Wait for a couple more seconds.
        sleep(2);

        puts("Closing session");

        // Gracefully close the connection.
        session_close(session, &error);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related Links

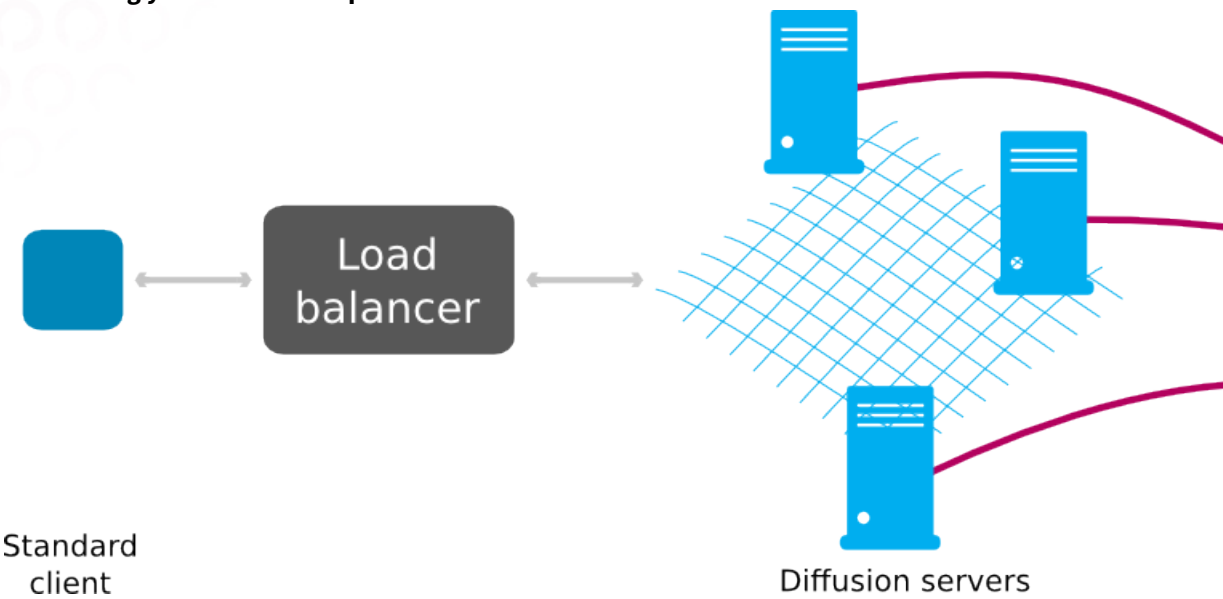
Features on page 169

Units of functionality within the Unified API are exposed as features.

Control features

Some features can be classified as control features. These control features contain capabilities that can be used to administer topics, clients or the Diffusion server.

Abstracting your control components



Using the control features of a client enables you to locate your control components outside of the Diffusion server. You can deploy your clients on separate systems to the Diffusion servers, in a separate data center, or in the cloud, enabling greater flexibility in your architecture.

Keeping your control components abstracted from Diffusion enables you to treat Diffusion as a messaging fabric. This supports scalable and cloud deployments.

TopicControl

Use the TopicControl feature to enable a client session to create and manage topics.

The TopicControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
TopicControl	YES	YES	YES	PARTIAL Does not support paged topics

Related Links

Features on page 169

Units of functionality within the Unified API are exposed as features.

Managing topics
A client can use the TopicControl feature of the Unified API to add and remove topics at the server.

Currently all topics created using a client have a lifespan the same as the Diffusion server. The topics remain at the Diffusion server even after the client session that created them has closed unless you explicitly specify that the topic is removed.

Adding new topics

Required permissions: modify_topic

For a client to create a topic it must first define the topic details that describe the topic. Builders of topic details can be created using the TopicControl feature. For more information, see Topic details on page 181.

You can use the same instance of topic details to create many topics. This is recommended when many topics with the same definition are to be created, because caching optimizations occur that prevent complex definitions from being transmitted to the Diffusion server many times.

For some types of topic, setting up metadata is part of the task of describing the topic.

The client can use the TopicControl feature to supply the initial state of the topic to the Diffusion server, as content, when the topic is created.

The `addTopic` operation is asynchronous and calls back to notify of either successful creation of the topic or failure to create the topic. If the topic add fails at the Diffusion server, the reason for failure is returned. Possible reasons for failure include the following:

- The topic already exists at the Diffusion server
- The name of the supplied topic is not valid
- The supplied details are not valid. This can occur only if properties are supplied.
- A user-supplied class cannot be found or instantiated. This can occur if you try to create a routing, paged, custom, or service topic and you have defined server-side classes to instantiate the topic.
- A referenced topic cannot be found
- Permission to create the topic was denied
- An error occurred trying to initialize the newly created topic with the supplied content, possibly because it was not validly formatted

A client can create topics subordinate to topics created by another client.

> **Note:** It is not currently possible to add new topics under branches of the topic tree that have been created by internal publishers.

Removing topics

Required permissions: modify_topic

A client can remove topics anywhere in the topic tree. The remove operation takes a topic selector, which enables the client to remove many topics at once. The removal of a topic also causes the removal of all topics beneath it in the tree.

DEPRECATED: Providing state for delegated topics

Required permissions: update_topic, register_handler

A client can register itself as a state provider for any branch of the topic tree.

When a client is registered as a state provider, all requests for the state of delegated topics in that branch are routed to the client, which returns content that defines the current topic state.

Creating paged topics

Use the `TopicDetails.Builder` classes with the TopicControl feature to create paged topics.

You can create a paged topic from the control client by using the following topic details builder interfaces:

`PagedRecordTopicDetails.Builder`

Use this builder to create a paged record topic.

`PagedStringTopicDetails.Builder`

Use this builder to create a paged string topic.

Paged topics can be ordered or unordered. You can define the ordering of a paged topic by using a user-defined comparator class that is located on the Diffusion server or by declaring the rules that are used for the ordering when you create the topic.

Creating an unordered paged topic

Use the `unordered` method of the topic details builder to set the paged topic to be unordered. By default paged topics are unordered, but you can also use this method to remove any ordering that had previously been specified for the paged topic.

In an unordered paged topic, the topic updates are added to the end of the paged topic in the order they are received.

Creating an ordered paged topic that uses a comparator

Use the `order` method of the topic details builder to set the paged topic to be ordered.

The `order` method can take the fully qualified name of a comparator class that is located on the class path of the Diffusion server, for example `com.example.comparators.MyComparator`. The comparator class must implement `java.util.Comparator<T>`, where *T* is either `Record` or `String` depending on the type of paged topic you are creating.

Creating an ordered paged record topic that uses declared rules

Use the `order` method of the paged record topic details builder to set the paged record topic to be ordered by one or more of its fields.

The `order` method can take one or more `OrderKey` objects.

An order key specifies the name of the field to order the lines in the paged record topic by. Because the order key uses a field name, you must define the fields that are in the paged record topic before you can specify which of these fields to use for ordering.

An order key can also, optionally, include an `Order` instance and a `Rules` instance. The `Order` instance defines the direction in which the lines are sorted and can be either DESCENDING or ASCENDING. If no order is defined, the default is ASCENDING. The `Rules` instance defines what additional ordering rules are applied to the ordering. The `Rules` instance can be of the type NO_RULES or the type COLLATION. If no rules are defined, the default is NO_RULES. You can use COLLATION-type rules to specify additional ordering rules by creating a `CollationRules` instance that defines these rules. The rules are defined using a String that uses the format defined by the Diffusion `java.text.RuleBasedCollator`. For more information, see RuleBasedCollator.

Supply the order keys to the `order` method in the order that they are to be applied. For example, if you specify an order key for the Name field then an order key for the Date field, lines are ordered first by the Name field and then, within those lines that have the same value in the Name field, by the value in the Date field.

Creating an ordered paged string topic that uses declared rules

Use the `order` method of the paged string topic details builder to set the paged string topic to be ordered.

The `order` method can take an `Order` instance and a `Rules` instance. If no parameters are passed to the `order` method, default values are used. The `Order` instance defines the direction in which the lines are sorted and can be either DESCENDING or ASCENDING. If no order is defined, the default is ASCENDING. The `Rules` instance defines what additional ordering rules are applied to the ordering. The `Rules` instance can be of the type NO_RULES or the type COLLATION. If no rules are defined, the default is NO_RULES. You can use COLLATION-type rules to specify additional ordering rules by creating a `CollationRules` instance that defines these rules. The rules are defined using a String that uses the format defined by the Diffusion `java.text.RuleBasedCollator`. For more information, see RuleBasedCollator.

Duplicate lines

You can also pass a duplicates policy into the `order` method, which defines where duplicate lines are inserted into the paged topic. The duplicates policy can have one of the following values:

FIRST

Insert the new line before the first matching duplicate.

LAST

Insert the new line after the last matching duplicate.

NOT_ALLOWED

Do not insert the duplicate and return an error.

If no duplicates policy is defined for an ordered paged topic, the default is NOT_ALLOWED and an error is returned if an attempt is made to add a duplicate line.

Creating a metadata definition for a record topic

You can use the Unified API to specify the metadata structure that describes the byte data content of a message.

About this task

Publishing clients define the metadata structure for messages. This metadata structure can be used when defining a topic. All messages placed on the topic must conform to the metadata structure.

The Unified API for the following platforms provides builder methods that enable you to define the metadata structure:

- JavaScript
- Java
- .NET

The C Unified API does not provide builder methods, but instead takes the definition of the metadata as XML. For more information, see Using the C Unified API on page 471.

The following example demonstrates how to define the metadata structure using the Java Unified API.

Note: Where there notation `c.p.d` is used in class or package names, it indicates `com.pushtechnology.diffusion`.

Procedure

1. Define the metadata structure.
   a) Import `c.p.d.client.Diffusion` and the following classes from the `c.p.d.client.content.metadata` package:
      - `MetadataFactory`
      - `MContent`
      - `MRecord`
      - `MField`
      - `MString`
      - `MIntegerString`
      - `MDecimalString`
      - `MCustomString`
   b) Use the `Diffusion.metadata` method to get a `MetadataFactory`.

```
private final MetadataFactory factory =
 Diffusion.metadata();
```

   c) Use the methods on the `MetadataFactory` to specify the content, record, and field definitions that make up the metadata structure.

      For example, the following code uses a content builder to create content metadata with a single record type that can occur zero to *n* times.

```
public MContent createContentRepeating() {
    return
        factory.contentBuilder("Content").
            add(
                factory.record(
                    "Rec1",
                    factory.string("A"),
                    factory.string("B")),
                0,
                -1).
            build();

}
```

      For more information, see Java Unified API documentation.
2. Create a record topic and apply the metadata definition to it.
   a) Import the TopicControl feature, `Session` class, and `RecordTopicDetails` class.

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicContr
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
```

   b) Create a `Session` instance and use it to get the TopicControl feature.

```
final Session session = Diffusion.sessions().open("ws://
diffusion.example.com:80");
final TopicControl tc =
 session.feature(TopicControl.class);
```

c) Get a topic builder for a record topic from the TopicControl feature.

```
final RecordTopicDetails.Builder builder =

 tc.newDetailsBuilder(RecordTopicDetails.Builder.class);
```

d) Use the `metadata` method of the topic builder to create the topic definition.

```
tc.addTopic(
            TOPIC_NAME,
            builder.metadata(metadata).build(),
            new TopicControl.AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    theTopic = topic;
                }

            });
```

Example: A client that creates a metadata definition and uses it when creating a topic.

```
package com.example.metadata;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MContent;
import
 com.pushtechnology.diffusion.client.content.metadata.MDecimalString;
import
 com.pushtechnology.diffusion.client.content.metadata.MField;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
import
 com.pushtechnology.diffusion.client.types.UpdateOptions;


/**
 * An example of a client creating metadata definition and
 using it when creating a
 * topic definition.
 */
public class ControlClient {

    private final MetadataFactory factory =
 Diffusion.metadata();

    /**
     * Creates control client instance.
     */
    public ControlClient() {

        // Create the Session
        final Session session = Diffusion.sessions()
```

```
                .open("ws://diffusion.example.com:80");

        // Add the TopicControl feature
        final TopicControl tc =
session.feature(TopicControl.class);

        // Create metadata for the topic
        final MContent metadata = defineMetadata();

        // Get a topic builder
        final RecordTopicDetails.Builder builder =

tc.newDetailsBuilder(RecordTopicDetails.Builder.class);

        // Create the topic with metadata
        tc.addTopic(
            TOPIC_NAME,
            builder.metadata(metadata).build(),
            new TopicControl.AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    theTopic = topic;
                }

            });

    }


    /**
     * A simple example of creating content metadata with two
records.
     *
     * @return content metadata
     */
    public MContent defineMetadata() {
        return factory.content(
            "Content",
            createNameAndAddressRecord(),
            createMultipleRateCurrencyRecord("Exchange
Rates", 5));
    }

    /**
     * Creates a simple name and address record with fixed
name single
     * multiplicity fields.
     *
     * @return record definition.
     */
    public MRecord createNameAndAddressRecord() {
        return factory.record(
            "NameAndAddress",
            factory.string("FirstName"),
            factory.string("Surname"),
            factory.string("HouseNumber"),
            factory.string("Street"),
            factory.string("Town"),
            factory.string("State"),
            factory.string("PostCode"));
    }

    /**
```

```
     * This creates a record with two fields, a string called
 "Currency" and a
     * decimal string called "Rate" with a default value of
 1.00 which repeats a
     * specified number of times.
     *
     * @param name the record name
     * @param occurs the number of occurrences of the "Rate"
 field
     * @return the metadata record
     */
    public MRecord createMultipleRateCurrencyRecord(String
 name, int occurs) {
        return factory.recordBuilder(name).
            add(factory.string("Currency")).
            add(factory.decimal("Rate", "1.00"), occurs).
            build();
    }

}
```

Deriving a single value topic data type or a record topic metadata definition from a value

When creating a record topic or single value topic, you can pass in a value that the Diffusion uses to derive the metadata definition for a record topic or the data type for a single value topic.

About this task

The Unified API for the following platforms allow you to provide a value when adding a record or single value topic from which the metadata structure or data type is derived:

- JavaScript
- Java
- .NET

> **Note:** Using this method to define the metadata structure for record topics has limitations.
>
> - Field types are derived from the values provided. Avoid providing initial values that are ambiguous. For example, if you intend that a field have a string data type, do not provide an initial value that is numeric. In this case a decimal or integer type is derived.
> - It is not possible for Diffusion to detect the intent to use repeating records or fields. If your metadata structure is to contain repeating records or fields, you cannot define your topic using a value. Instead create your metadata definition explicitly, using the provided builder methods.
> - In some APIs (for example, JavaScript you must have defined the metadata in order to create record content. In many cases, it is better when adding a topic to use this metadata definition in addition to providing an initial value.

Procedure

1. Add a topic, providing a value to the method from which Diffusion can derive the metadata structure or data type of the topic.

   JavaScript

   ```
   // Create a topic with string values, and an initial value of
    "xyz".
   session.topics.add('topic/string', 'xyz');
   ```

```
// Create a topic with integer values, and an initial value of
 123.
session.topics.add('topic/integer', 123);

// Create a topic with decimal values, with an implicit scale
 of 2 and an initial value of 1.23.
session.topics.add('topic/decimal', 1.23);

// Create record content from previously defined metadata
var builder = metadata.builder();

// Values must be set before a value can be created
builder.add('game', { title : 'Planet Express!', count : 3 });

builder.add('player', { name : 'Fry', score : 0 });
builder.add('player', { name : 'Amy', score : 0 });
builder.add('player', { name : 'Kif', score : 0 });

// Build a content instance
var initial_value = builder.build();

// Create a topic with record content. The metadata structure
 is derived from the structure of the initial value.
session.topics.add('games/game',
 initial_value).then(function() {
   console.log('Topic was added with initial value');
});
```

Java

```

// Create a topic with string values, and an initial value of
 "xyz".
topicControl.addTopicFromValue( "topic/string", "xyz",
 callback);

// Create a topic with integer values, and an initial value of
 42.
topicControl.addTopicFromValue( "topic/integer", "42",
 callback);

// Create a topic with decimal values, with an implicit scale
 of 3 and an initial value of 2.718.
topicControl.addTopicFromValue( "topic/decimal", "2.718",
 callback);

// Create a topic with record content. Use
 RecordCOntentBuilder and its methods to construct the
 content.
// The metadata structure is derived from the structure of the
 initial value.
topicControl.addTopicFromValue( "topic/record",

 Diffusion.content().newBuilder(RecordContentBuilder.class)

 .putFields(initialValues).build(),
                                    callback);
```

.NET

```
var topicControl = session.GetTopicControlFeature();
```

```
var callback = new TopicControlAddCallbackDefault();

// Create a topic with string values and an initial value of
 'xyz'.
topicControl.AddTopicFromValue( "topic/string", "xyz",
 callback );

// Create a topic with integer values and an initial value of
 42.
topicControl.AddTopicFromValue( "topic/integer", "42",
 callback );

// Create a topic with decimal values with an implicit scale
 of 3 and an initial value of 2.718.
topicControl.AddTopicFromValue( "topic/decimal", "2.718",
 callback );

// Create a topic with record content.  Use
 IRecordContentBuilder and its methods to construct the
 content.
// The metadata structure is derived from the structure of the
 initial value.
var initialValues = new[]{"1","2","3"};

topicControl.AddTopicFromValue(
 "topic/record",
 Diffusion.Content.NewBuilder<IRecordContentBuilder>().PutFields( initial
 callback );
```

For more information about the format of the initial value in each API and how metadata structure and data types are derived from that value, see the API documentation for that API.

2. Diffusion derives the structure of the metadata or the data type for the topic from the provided initial value.

   This metadata structure or data type must be adhered to by all subsequent updates to the topic.

3. The value used to create the topic is set as the initial value of the topic and sent to any subscribed clients.

Handling subscriptions to missing topics

A client can use the TopicControl feature of the Unified API to handle subscription or fetch requests for topics that do not exist.

Registering a missing topic handler and dynamically adding topics

Required permissions: modify_topic, register_handler

You can use the TopicControl feature to dynamically create topics on demand when a client tries to subscribe or fetch a topic that does not exist.

The client can register itself as a handler for missing topics for any part of the topic tree. The client is notified of attempts to subscribe to or fetch topics that are subordinate to that topic and that do not exist. This enables the client to create the topics and notify the Diffusion server that the client operation subscribe or fetch can proceed.

> **Note:** The handler is called only when a client attempts to subscribe or fetch using a single topic path. If another type of selector is used to subscribe to or fetch the topic, the handler is not notified.

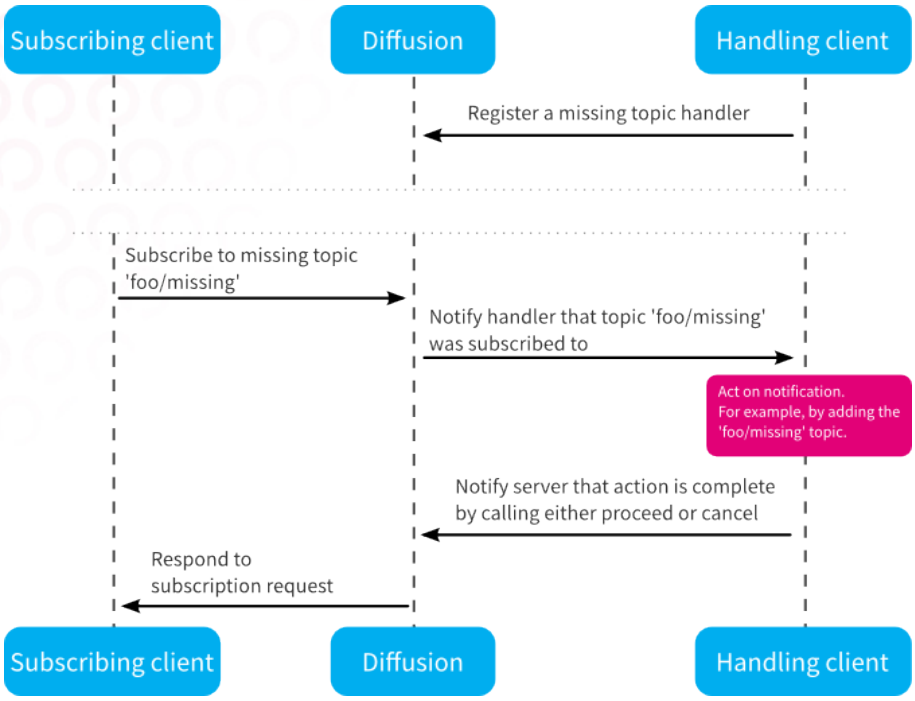

Subscribing client
Diffusion
Handling client
Register a missing topic handler
Subscribe to missing topic 'foo/missing'
Notify handler that topic 'foo/missing' was subscribed to
Act on notification. For example, by adding the 'foo/missing' topic.
Notify server that action is complete by calling either proceed or cancel
Respond to subscription request
Subscribing client
Diffusion
Handling client

Figure 40: Flow from a subscribing client to the client that handles a missing topic subscription

Removing topics with sessions

A client can use the TopicControl feature of the Unified API to specify that the Diffusion server removes a topic or topics after the client session closes or fails.

Required permissions: modify_topic

When a client registers a request that a branch of the topic tree be removed when its session closes the following events occur:

1. The client registers the removal request on a branch of the topic tree and passes in a topic tree handler.
   - The removal request is registered against a topic path. This is a path that identifies a branch of the topic tree, for example foo/bar. The removal request is registered for the branch of the topic tree, for example the topics foo/bar/baz and foo/bar/fred/qux are included in the specified branch of the topic tree.
   - You cannot register a removal request above or below an existing removal request. For example, if a client has registered a removal request against foo/bar/fred another client cannot register a removal request against foo/bar or foo/bar/fred/qux.
2. The server validates the request and gives one of the following responses:
   - If the request is not valid, the Diffusion server calls the `onError` callback of the topic tree handler.

     For example, a registration request is not valid if it registers against a topic branch that is above or below a branch where an existing removal request is registered.
   - If the request is valid, the Diffusion server calls the `OnActive` callback of the topic tree handler and provides a `Registration` object to the client.
3. If the client wants to deregister a removal request, it can call the `onClose` method of the `Registration` object for that removal request.

4. Other clients can register removal requests against a topic that already has a removal request registered against it. For example, if one session on the Diffusion server has a removal request registered against foo/bar/baz, another session on the Diffusion server can also register a removal request against foo/bar/baz.
5. When a client session closes or fails, if it has registered removal requests one of the following things happens:
   - If there are still open sessions that have removal requests for that branch of the topic tree, the Diffusion server takes no action.
   - If there are no open sessions that have removal requests for that branch of the topic tree, the Diffusion server removes all topics in that branch of the topic tree.

Removal requests and topic replication

If all sessions on a Diffusion server that have a remove topics removal request for a branch of the topic tree close, the topics are removed even if that topic is replicated and sessions on other Diffusion servers have removal requests registered against that part of the tree. When the topics are removed on the server, that change is replicated to all other servers that participate in replication for these topics.

Example: Create a topic
The following examples use the TopicControl feature in the Unified API to create topics.

JavaScript

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. Topics can be created with a specified topic path and
 value. If the path contains multiple levels, any
    // intermediary topics that do not exist will be created
 automatically with a stateless topic type.

    // Create a topic with string values, and an initial value
 of "xyz".
    session.topics.add('topic/string', 'xyz');

    // Create a topic with integer values, and an initial value
 of 123.
    session.topics.add('topic/integer', 123);

    // Create a topic with decimal values, with an implicit
 scale of 2 and an initial value of 1.23.
    session.topics.add('topic/decimal', 1.23);

    // 2. Adding a topic returns a result, which allows us to
 handle when the operation has either
    // completed successfully or encountered an error.
    session.topics.add('topic/result',
'abc').then(function(result) {
        console.log('Added topic: ' + result.topic);
    }, function(reason) {
        console.log('Failed to add topic: ', reason);
    });

    // Adding a topic that already exists will succeed, so long
 as it has the same value type
```

```
   session.topics.add('topic/result',
'xyz').then(function(result) {
       // result.added will be false, as the topic already
existed
       console.log('Added topic: ' + result.topic,
result.added);
   });

   // Because the result returned from adding a topic is a
promise, we can easily chain
   // multiple topic adds together
   session.topics.add('chain/foo',
1).then(session.topics.add('chain/bar', 2))

.then(session.topics.add('chain/baz', 3))

.then(session.topics.add('chain/bob', 4))
                                       .then(function() {
                                          console.log('Added all
topics');
                                       }, function(reason) {
                                          console.log('Failed to
add topic', reason);
                                       });

   // 3. Metadata can be used to create topics that will
contain values of a specified format.

   // RecordContent formats data in a series of records and
fields, similar to tabular data.
   // Each record & field is named, allowing direct lookup of
values. Each field value has a
   // particular type (string, integer, decimal)
   var metadata = new diffusion.metadata.RecordContent();

   // Records are like rows in a table. They can have fields
assigned, with default values.
   // You can add fields all at once like this, or individually
(see below).
   var game = metadata.addRecord('game', 1, {
       'title' : metadata.string(),
       'round' : metadata.integer(0),
       'count' : metadata.integer(0)
   });

   // Records and fields can be set as occurring a certain
number of times.
   var player = metadata.addRecord('player', metadata.occurs(0,
8));

   // Add fields to a record individually.
   player.addField('name', 'Anonymous');
   player.addField('score', 0);

   // Adding the topic works just like normal.
   session.topics.add('games/some-game', metadata);

   // And the metadata can be re-used for multiple topics.
   session.topics.add('games/some-other-game', metadata);

   // 4. Using metadata, it is possible to create a topic with
both a metadata format, and the initial value
```

```
    // Topic values can be produced from metadata via the
 builder interface
    var builder = metadata.builder();

    // Values must be set before a value can be created
    builder.add('game', { title : 'Planet Express!', count :
 3 });

    builder.add('player', { name : 'Fry', score : 0 });
    builder.add('player', { name : 'Amy', score : 0 });
    builder.add('player', { name : 'Kif', score : 0 });

    // Build a content instance
    var content = builder.build();

    // Now that the content has been built, a topic can be added
 with the metadata & initial value
    session.topics.add('games/yet-another-game', metadata,
 content).then(function() {
        console.log('Topic was added with metadata and
 content');
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.List;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.callbacks.TopicTreeHandler;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.content.RecordContentBuilder;
import
 com.pushtechnology.diffusion.client.content.metadata.MContent;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
import
 com.pushtechnology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client to add topics.
 * <P>
 * This uses the 'TopicControl' feature only.
 * <P>
 * To add or remove topics, the client session must have the
 'modify_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
```

```
public class ControlClientAddingTopics {

    private final Session session;

    private final TopicControl topicControl;

    /**
     * Constructor.
     */
    public ControlClientAddingTopics() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);

    }

    /**
     * Adds a topic with type derived from value.
     * <P>
     * This uses the simple convenience method for adding topics
 where the topic
     * type and metadata are derived from a supplied value which
 can be any
     * object. For example, an Integer would result in a single
 value topic of
     * type integer.
     *
     * @param topicPath full topic path
     * @param initialValue an optional initial value for the
 topic
     * @param context this will be passed back to the callback
 when reporting
     *        success or failure of the topic add
     * @param callback to notify result of operation
     * @param <T> the value type
     * @return the topic details used to add the topic
     */
    public <T> TopicDetails addTopicForValue(
        String topicPath,
        T initialValue,
        String context,
        AddContextCallback<String> callback) {

        return topicControl.addTopicFromValue(
            topicPath,
            initialValue,
            context,
            callback);
    }

    /**
     * Add a record topic from a list of initial values.
     * <P>
     * This demonstrates the simplest mechanism for adding a
 record topic by
     * supplying values that both the metadata and the initial
 values are
     * derived from.
     *
```

```
     * @param topicPath full topic path
     * @param initialValues the initial values for the topic
fields which will
     *        also be used to derive the metadata definition of
the topic
     * @param context this will be passed back to the callback
when reporting
     *        success or failure of the topic add
     * @param callback to notify result of operation
     * @return the topic details used to add the topic
     */
    public TopicDetails addRecordTopic(
        String topicPath,
        List<String> initialValues,
        String context,
        AddContextCallback<String> callback) {

        return topicControl.addTopicFromValue(
            topicPath,

Diffusion.content().newBuilder(RecordContentBuilder.class)
                .putFields(initialValues).build(),
            context,
            callback);

    }

    /**
     * Adds a record topic with supplied metadata and optional
initial content.
     * <P>
     * This example shows details being created and would be
fine when creating
     * topics that are all different but if creating many record
topics with the
     * same details then it is far more efficient to pre-create
the details.
     *
     * @param topicPath the full topic path
     * @param metadata pre-created record metadata
     * @param initialValue optional initial value for the topic
which must have
     *        been created to match the supplied metadata
     * @param context context passed back to callback when topic
created
     * @param callback to notify result of operation
     */
    public void addRecordTopic(
        String topicPath,
        MContent metadata,
        Content initialValue,
        String context,
        AddContextCallback<String> callback) {

        final TopicDetails details =

topicControl.newDetailsBuilder(RecordTopicDetails.Builder.class)
                .metadata(metadata).build();

        topicControl.addTopic(
            topicPath,
            details,
            initialValue,
```

```
            context,
            callback);
    }

    /**
     * Remove a single topic given its path.
     *
     * @param topicPath the topic path
     * @param callback notifies result of operation
     */
    public void removeTopic(String topicPath, RemoveCallback
callback) {
        topicControl.removeTopics(
            ">" + topicPath, // convert to a topic path selector
            callback);
    }

    /**
     * Remove one or more topics using a topic selector
expression.
     *
     * @param topicSelector the selector expression
     * @param callback notifies result of operation
     */
    public void removeTopics(String topicSelector,
RemoveCallback callback) {
        topicControl.removeTopics(
            topicSelector,
            callback);
    }

    /**
     * Request that the topic {@code topicPath} and its
descendants be removed
     * when the session is closed (either explicitly using
{@link Session#close}
     * , or by the server). If more than one session calls this
method for the
     * same {@code topicPath}, the topics will be removed when
the last session
     * is closed.
     *
     * <p>
     * Different sessions may call this method for the same
topic path, but not
     * for topic paths above or below existing registrations on
the same branch
     * of the topic tree.
     *
     * @param topicPath the part of the topic tree to remove
when the last
     *        session is closed
     */
    public void removeTopicsWithSession(String topicPath) {
        topicControl.removeTopicsWithSession(
            topicPath, new TopicTreeHandler.Default());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
```

```
    }

}
```

.NET

```
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Content.Metadata;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client to add topics.
 ///
 /// This uses the <see cref="ITopicControl"/> feature only.
 ///
 /// To add or remove topics, the client session must have the
 <see cref="TopicPermission.MODIFY_TOPIC"/> permission
 /// for that branch of the topic tree.
 /// </summary>
 public class ControlClientAddingTopics
 {
  #region Fields

  private readonly ISession session;
  private readonly ITopicControl topicControl;

  #endregion Fields

  #region Constructor

  public ControlClientAddingTopics()
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topicControl = session.GetTopicControlFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Adds a topic with the type derived from the value.
  ///
  /// This uses the simple convenience method for adding topics
 where the topic type and metadata are derived
  /// from a supplied value which can be any object.  For
 example, an integer would result in a single value topic
  /// of type 'integer'.
  /// </summary>
  /// <typeparam name="T">The value type.</typeparam>
```

```
  /// <param name="topicPath">The full topic path.</param>
  /// <param name="initialValue">An optional initial value for
 the topic.</param>
  /// <param name="context">This will be passed back to the
 callback when reporting success or failure of the
  /// topic add.</param>
  /// <param name="callback">To notify the result of the
 operation.</param>
  /// <returns>The topic details used to add the topic.</
returns>
  public ITopicDetails AddTopicForValue<T>(
   string topicPath,
   T initialValue,
   string context,
   ITopicControlAddContextCallback<string> callback )
  {
   return topicControl.AddTopicFromValue( topicPath,
 initialValue, context, callback );
  }

  /// <summary>
  /// Add a record topic from a list of initial values.
  ///
  /// This demonstrates the simplest mechanism for adding a
 record topic by supplying values that both the
  /// metadata and the initial values are derived from.
  /// </summary>
  /// <param name="topicPath">The full topic path.</param>
  /// <param name="initialValues">The initial values for the
 topic fields which will also be used to derive the
  /// metadata definition of the topic.</param>
  /// <param name="context">This will be passed back to the
 callback when reporting success or failure of the
  /// topic add.</param>
  /// <param name="callback">To notify the result of the
 operation.</param>
  /// <returns></returns>
  public ITopicDetails AddRecordTopic(
   string topicPath,
   List<string> initialValues,
   string context,
   ITopicControlAddContextCallback<string> callback )
  {
   return topicControl.AddTopicFromValue(
    topicPath,

 Diffusion.Content.NewBuilder<IRecordContentBuilder>().PutFields( initialVa
    context,
    callback );
  }

  /// <summary>
  /// Adds a record topic with supplied metadata and optional
 initial content.
  ///
  /// This example shows details being created and would be fine
 when creating topics that are all different, but
  /// if creating many record topics with the same details, then
 it is far more efficient to pre-create the
  /// details.
  /// </summary>
  /// <param name="topicPath">The full topic path.</param>
```

```
  /// <param name="metadata">The pre-created record metadata.</
param>
  /// <param name="initialValue">The optional initial value for
 the topic which must have been created to match
  /// the supplied metadata.</param>
  /// <param name="context">The context passed back to the
 callback when the topic is created.</param>
  /// <param name="callback">To notify the result of the
 operation.</param>
  public void AddRecordTopic(
   string topicPath,
   IMContent metadata,
   IContent initialValue,
   string context,
   ITopicControlAddContextCallback<string> callback )
  {
   var details =
 topicControl.CreateDetailsBuilder<IRecordTopicDetailsBuilder>().Metadata(

   topicControl.AddTopic( topicPath, details, initialValue,
 context, callback );
  }

  /// <summary>
  /// Remove a single topic given its path.
  /// </summary>
  /// <param name="topicPath">The topic path.</param>
  /// <param name="callback">Notifies the result of the
 operation.</param>
  public void RemoveTopic( string topicPath,
 ITopicControlRemoveCallback callback )
  {
   topicControl.RemoveTopics( ">" + topicPath, callback );
  }

  /// <summary>
  /// Remove one or more topics using a topic selector
 expression.
  /// </summary>
  /// <param name="topicSelector">The selector expression.</
param>
  /// <param name="callback">Notifies the result of the
 operation.</param>
  public void RemoveTopics( string topicSelector,
 ITopicControlRemoveCallback callback )
  {
   topicControl.RemoveTopics( topicSelector, callback );
  }

  /// <summary>
  /// Request that the topic path and its descendants be removed
 when the session is closed (either explicitly
  /// using <see cref="ISession.Close"/>, or by the server).  If
 more than one session calls this method for the
  /// same topic path, the topics will be removed when the last
 session is closed.
  ///
  /// Different sessions may call this method for the same topic
 path, but not for topic paths above or below
  /// existing registrations on the same branch of the topic
 tree.
  /// </summary>
```

```
  /// <param name="topicPath">The part of the topic tree to
 remove when the last session is closed.</param>
  public void RemoveTopicsWithSession( string topicPath )
  {
   topicControl.RemoveTopicsWithSession( topicPath, new
 DefaultTopicTreeHandler() );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "service/svc-topic-control.h"
#include "utils.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

// Various handlers which are common to all add_topic()
 functions.
static int
on_topic_added(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 puts("on_topic_added");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
```

```
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 printf("on_topic_add_failed: %d\n", response->response_code);
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 puts("on_topic_add_discard");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_removed(SESSION_T *session, const
 SVC_REMOVE_TOPICS_RESPONSE_T *response, void *context)
{
 puts("on_topic_removed");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_remove_discard(SESSION_T *session, const
 SVC_REMOVE_TOPICS_RESPONSE_T *response, void *context)
{
 puts("on_topic_remove_discard");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}
/*
 *
 */
int main(int argc, char** argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }

 // Setup for condition variable
 apr_initialize();
 apr_pool_create(&pool, NULL);
 apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_DEFAULT,
 pool);
 apr_thread_cond_create(&cond, pool);
```

```
 // Setup for session
 SESSION_T *session;
 DIFFUSION_ERROR_T error;
 session = session_create(url, principal, credentials, NULL,
 NULL, &error);
 if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        // Common params for all add_topic() functions.
        ADD_TOPIC_PARAMS_T common_params = {
                .on_topic_added = on_topic_added,
                .on_topic_add_failed = on_topic_add_failed,
                .on_discard = on_topic_add_discard
        };

 // Stateless topic details
 TOPIC_DETAILS_T *topic_details =
 create_topic_details_stateless();
        ADD_TOPIC_PARAMS_T stateless_params = common_params;
        stateless_params.topic_path = "stateless";
        stateless_params.details = topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, stateless_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value string data, no default data.
 TOPIC_DETAILS_T *string_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_STRING);
        ADD_TOPIC_PARAMS_T string_params = common_params;
        string_params.topic_path = "string";
        string_params.details = string_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, string_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value string data, with default data.
        ADD_TOPIC_PARAMS_T string_data_params = common_params;
        string_data_params.topic_path = "string-data";
        string_data_params.details = string_topic_details;
 BUF_T *sample_data_buf = buf_create();
 buf_write_string(sample_data_buf, "Hello, world");
        string_data_params.content =
 content_create(CONTENT_ENCODING_NONE, sample_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, string_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value integer data.
        TOPIC_DETAILS_T *integer_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_INTEGER_STRING);
 integer_topic_details-
>topic_details_params.integer.default_value = 999;
```

```
        ADD_TOPIC_PARAMS_T integer_params = common_params;
        integer_params.topic_path = "integer";
        integer_params.details = integer_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, integer_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value integer data with content.
        ADD_TOPIC_PARAMS_T integer_data_params = common_params;
        integer_data_params.topic_path = "integer-data";
        integer_data_params.details = integer_topic_details;
 BUF_T *integer_data_buf = buf_create();
 buf_sprintf(integer_data_buf, "%d", 123);
        integer_data_params.content =
 content_create(CONTENT_ENCODING_NONE, integer_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, integer_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value decimal data.
 TOPIC_DETAILS_T *decimal_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_DECIMAL_STRING);
 decimal_topic_details-
>topic_details_params.decimal.default_value = 123.456;
 decimal_topic_details->topic_details_params.decimal.scale = 4;

        ADD_TOPIC_PARAMS_T decimal_params = common_params;
        decimal_params.topic_path = "decimal";
        decimal_params.details = decimal_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, decimal_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value decimal data with content.
        ADD_TOPIC_PARAMS_T decimal_data_params = common_params;
        decimal_data_params.topic_path = "decimal-data";
        decimal_data_params.details = decimal_topic_details;
 BUF_T *decimal_data_buf = buf_create();
 buf_sprintf(decimal_data_buf, "%f", 987.654);
 decimal_data_params.content =
 content_create(CONTENT_ENCODING_NONE, decimal_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, decimal_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Manually specify schema for single value string topic data.
 BUF_T *manual_schema = buf_create();
 buf_write_string(manual_schema,
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>
\n");
 buf_write_string(manual_schema,
  "<field name=\"x\" type=\"string\" default=\"xyzzy\"
 allowsEmpty=\"true\"/>");
 TOPIC_DETAILS_T *manual_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_STRING);
```

```
 manual_topic_details->user_defined_schema = manual_schema;

        ADD_TOPIC_PARAMS_T string_manual_params = common_params;
        string_manual_params.topic_path = "string-manual";
        string_manual_params.details = manual_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, string_manual_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Simple record/field structure, created by manually
 specifying the
 // schema.
 BUF_T *record_schema = buf_create();
 buf_write_string(record_schema,
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?
>");
 buf_write_string(record_schema,
  "<message topicDataType=\"record\" name=\"MyContent\">");
 buf_write_string(record_schema,
  "<record name=\"Record1\">");
 buf_write_string(record_schema,
  "<field type=\"string\" default=\"\" allowsEmpty=\"true\"
 name=\"Field1\"/>");
 buf_write_string(record_schema,
  "<field type=\"integerString\" default=\"0\" allowsEmpty=
\"false\" name=\"Field2\"/>");
 buf_write_string(record_schema,
  "<field type=\"decimalString\" default=\"0.00\" scale=\"2\"
 allowsEmpty=\"false\" name=\"Field3\"/>");
 buf_write_string(record_schema,
  "</record>");
 buf_write_string(record_schema,
  "</message>");
 TOPIC_DETAILS_T *record_topic_details =
 create_topic_details_record();
 record_topic_details->user_defined_schema = record_schema;

        ADD_TOPIC_PARAMS_T record_params = common_params;
        record_params.topic_path = "record";
        record_params.details = record_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, record_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Remove topic tests
 puts("Adding topics remove_me/1 and remove_me/2");
 apr_thread_mutex_lock(mutex);
        ADD_TOPIC_PARAMS_T topic_params = common_params;
        topic_params.details = topic_details;

        topic_params.topic_path = "remove_me/1";
 add_topic(session, topic_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

        topic_params.topic_path = "remove_me/2";
 add_topic(session, topic_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);
```

```
 puts("Removing topics in 5 seconds...");
 sleep(5);

        REMOVE_TOPICS_PARAMS_T remove_params = {
                .on_removed = on_topic_removed,
                .on_discard = on_topic_remove_discard,
                .topic_selector = ">remove_me"
        };

 apr_thread_mutex_lock(mutex);
 remove_topics(session, remove_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 return(EXIT_SUCCESS);
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Create a paged topic
The following examples use the TopicControl feature in the Unified API to create paged topics.

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.Collection;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.content.update.PagedRecordOrderedUpdat
import
 com.pushtechnology.diffusion.client.content.update.PagedStringUnorderedUpd
import
 com.pushtechnology.diffusion.client.content.update.Update;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
```

```
 * An example of using a control client to create and update
 paged topics.
 * <P>
 * This uses the 'TopicControl' feature to create a paged topic
 and the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * This demonstrates some simple examples of paged topic updates
 but not all of
 * the possible ways in which they can be done.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientUpdatingPagedTopics {

    private static final String ORDERED_TOPIC = "Paged/Ordered";
    private static final String UNORDERED_TOPIC = "Paged/
Unordered";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final PagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
    private final PagedStringUnorderedUpdateFactory
 unorderedUpdateFactory;
    private Updater pagedUpdater = null;

    /**
     * Constructor.
     */
    public ControlClientUpdatingPagedTopics() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl =
 session.feature(TopicUpdateControl.class);

        orderedUpdateFactory =
            updateControl

 .updateFactory(PagedRecordOrderedUpdateFactory.class);
        unorderedUpdateFactory =
            updateControl

 .updateFactory(PagedStringUnorderedUpdateFactory.class);

        final MetadataFactory metadata = Diffusion.metadata();

        // Create an unordered paged string topic
        topicControl.addTopic(
            UNORDERED_TOPIC,
            topicControl.newDetails(TopicType.PAGED_STRING),
            new AddCallback.Default());
```

```
        // Create an ordered paged record topic
        final MRecord recordMetadata =
            metadata.record(
                "Record",
                metadata.string("Name"),
                metadata.string("Address"));

        topicControl.addTopic(
            ORDERED_TOPIC,
            topicControl.

newDetailsBuilder(PagedRecordTopicDetails.Builder.class).
                metadata(recordMetadata).
                order(new OrderKey("Name")).build(),
            new AddCallback.Default());

        // Register as updater for topics under the 'Paged'
branch
        updateControl.registerUpdateSource(
            "Paged",
            new UpdateSource.Default() {
                @Override
                public void onActive(String topicPath, Updater
updater) {
                    pagedUpdater = updater;
                }
            });
    }

    /**
     * Add a new line to the ordered topic.
     *
     * @param name the name field value
     * @param address the address field value
     * @param callback to notify result
     */
    public void addOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.add(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Update a line of an ordered topic.
     *
     * @param name the name of the line to update
     * @param address the new address field value
     * @param callback to notify result
     */
    public void updateOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.update(
```

```
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Remove a line from an ordered topic.
     *
     * @param name the name of the line to remove
     * @param callback to notify result
     */
    public void removeOrdered(String name, UpdateCallback
callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.remove(
                Diffusion.content().newRecord(name, "")),
            callback);
    }

    /**
     * Add a line or lines to the end of an unordered topic.
     *
     * @param values lines to add
     * @param callback to notify result
     */
    public void addUnordered(
        Collection<String> values,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.add(values),
            callback);
    }

    /**
     * Insert a line or lines at a specified index within an
unordered topic.
     *
     * @param index the index to add at
     * @param values lines to insert
     * @param callback to notify result
     */
    public void insertUnordered(
        int index,
        Collection<String> values,
        UpdateCallback callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.insert(index, values),
            callback);
    }

    /**
     * Update a line within an unordered topic.
     *
     * @param index the index of the line to update
     * @param value the new line value
     * @param callback to notify result
     */
    public void updateUnordered(
        int index,
```

```
        String value,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.update(index, value),
            callback);
    }

    /**
     * Remove a specific line from an unordered topic.
     *
     * @param index the line to remove
     * @param callback to notify result
     */
    public void removeUnordered(int index, UpdateCallback
 callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.remove(index),
            callback);
    }

    private void update(String topic, Update update,
 UpdateCallback callback)
        throws IllegalStateException {
        if (pagedUpdater == null) {
            throw new IllegalStateException("No updater");
        }
        pagedUpdater.update(topic, update, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">Paged",
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
}
```

.NET

```
using System;
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
```

```
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Topics.Update;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client to create and update
 paged topics.
 ///
 /// This uses the <see cref="ITopicControl"/> feature to create
 a paged topic and the <see cref="ITopicUpdateControl"/>
 /// feature to send updates to it.
 ///
 /// This demonstrates some simple examples of paged topic
 updates but not all of the possible ways in which they
 /// can be done.
 ///
 /// To send updates to a topic, the client session requires the
 UPDATE_TOPIC permission for that branch of the
 /// topic tree.
 /// </summary>
 public class ControlClientUpdatingPagedTopics
 {
  #region Fields

  private const string OrderedTopic = "Paged/Ordered";
  private const string UnorderedTopic = "Paged/Unordered";

  private readonly ISession session;
  private readonly ITopicControl topicControl;
  private readonly IPagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
  private readonly IPagedStringUnorderedUpdateFactory
 unorderedUpdateFactory;
  private static ITopicUpdater _pagedUpdater;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  public ControlClientUpdatingPagedTopics()
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topicControl = session.GetTopicControlFeature();
   var updateControl = session.GetTopicUpdateControlFeature();

   orderedUpdateFactory =
 updateControl.UpdateFactory<IPagedRecordOrderedUpdateFactory>();
   unorderedUpdateFactory =
 updateControl.UpdateFactory<IPagedStringUnorderedUpdateFactory>();

   var metadata = Diffusion.Metadata;

   // Create an unordered paged string topic
   topicControl.AddTopic(
    UnorderedTopic,
    topicControl.NewDetails( TopicType.PAGED_STRING ),
```

```
    new TopicControlAddCallbackDefault() );

  // Create an ordered paged record topic
  var recordMetadata = metadata.Record(
   "Record",
   metadata.String( "Name" ),
   metadata.String( "Address" ) );

  topicControl.AddTopic(
   OrderedTopic,
   topicControl
    .CreateDetailsBuilder<IPagedRecordTopicDetailsBuilder>()
    .Metadata( recordMetadata )
    .Order( new PagedRecordOrderKey( "Name" ) ).Build(),
   new TopicControlAddCallbackDefault() );

  // Register an updater for topics under the 'Paged' branch
  updateControl.RegisterUpdateSource( "Paged", new
UpdateSource() );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Add a new line from an ordered topic.
 /// </summary>
 /// <param name="name">The name field value.</param>
 /// <param name="address">The address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void AddOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
   orderedUpdateFactory.Add( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Update a line of an ordered topic.
 /// </summary>
 /// <param name="name">The name of the line to update.</param>
 /// <param name="address">The new address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void UpdateOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,

orderedUpdateFactory.Update( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Remove a line from an ordered topic.
 /// </summary>
```

```
  /// <param name="name">The name of the line to remove.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveOrdered( string name,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,

 orderedUpdateFactory.Remove( Diffusion.Content.NewRecord( name,
 "" ) ),
    callback );
  }

  /// <summary>
  /// Add a line or lines to the end of an unordered topic.
  /// </summary>
  /// <param name="values">The lines to add.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void AddUnordered( ICollection<string> values,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Add( values ),
    callback );
  }

  /// <summary>
  /// Insert a line or lines at a specified index within an
 unordered topic.
  /// </summary>
  /// <param name="index">The index at which to add the line.</
param>
  /// <param name="values">The lines to insert.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void InsertUnordered( int index, ICollection<string>
 values, ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Insert( index, values ),
    callback );
  }

  /// <summary>
  /// Remove a specific line from an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to remove.</
param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveUnordered( int index,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Remove( index ),
    callback );

  }
```

```
  /// <summary>
  /// Update a line within an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to update.</
param>
  /// <param name="value">The new line value.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void UpdateUnordered( int index, string value,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,
    unorderedUpdateFactory.Update( index, value ),
    callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   // Remove our topics and close the session when done.
   topicControl.RemoveTopics( ">Paged", new
 RemoveCallback( session ) );
  }

  #endregion Public Methods

  #region Private Methods

  private static void Update( string topic, IUpdate update,
 ITopicUpdaterUpdateCallback callback )
  {
   if( _pagedUpdater == null )
   {
    throw new InvalidOperationException( "The paged updater has
 not been initialised." );
   }

   _pagedUpdater.Update( topic, update, callback );
  }

  #endregion Private Methods

  #region Private Classes

  private class RemoveCallback :
 TopicControlRemoveCallbackDefault
  {
   #region Fields

   private readonly ISession theSession;

   #endregion Fields

   #region Constructor

   public RemoveCallback( ISession session )
   {
    theSession = session;
   }
```

```
    #endregion Constructor

    #region Overrides

    /// <summary>
    /// Notification that a call context was closed prematurely,
  typically due to a timeout or the session being
    /// closed.  No further calls will be made for the context.
    /// </summary>
    public override void OnDiscard()
    {
     theSession.Close();
    }

    /// <summary>
    /// Topic(s) have been removed.
    /// </summary>
    public override void OnTopicsRemoved()
    {
     theSession.Close();
    }

    #endregion Overrides
   }

   private class UpdateSource : TopicUpdateSourceDefault
   {
    #region Overrides

    /// <summary>
    /// State notification that this source is now active for the
  specified topic path, and is therefore in a valid
    /// state to send updates on topics at or below the
  registered topic path.
    /// </summary>
    /// <param name="topicPath">The registration path.</param>
    /// <param name="updater">An updater that may be used to
  update topics at or below the registered path.</param>
    public override void OnActive( string topicPath,
  ITopicUpdater updater )
    {
     _pagedUpdater = updater;
    }

    #endregion Overrides
   }

   #endregion Private Classes
  }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Receive missing topic notifications
The following examples use the TopicControl feature in the Unified API to register a missing topic notification handler.

Java

```
import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.RegisteredHandler;
```

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicAddFailRe
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.M
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.M
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.TopicDetails;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of registering a missing topic notification
 handler and processing
 * notifications using a control client.
 *
 * @author Push Technology Limited
 */
public final class ControlClientHandlingMissingTopicNotification
 {

    // UCI features
    private final Session session;
    private final TopicControl topicControl;
    private final TopicDetails details;

    /**
     * Constructor.
     */
    public ControlClientHandlingMissingTopicNotification() {
        // Create a session
        session =
 Diffusion.sessions().password("password").principal("admin").open("dpt://
localhost:8081");

        topicControl = session.feature(TopicControl.class);

        // Registers a missing topic notification on a topic
 path
        topicControl.addMissingTopicHandler("A", new
 MissingTopicNotificationHandler());

        // For details that may be reused many times it is far
 more efficient to
        // create just once - this creates a default string type
 details.
        details =
 topicControl.newDetails(TopicType.SINGLE_VALUE);

    }

    // Private class that implements MissingTopicHandler
 interface
    private final class MissingTopicNotificationHandler
 implements
            MissingTopicHandler {
        /**
         * @param topicPath
         *            - the path that the handler is active for
         * @param registeredHandler
```

```
         *            - allows the handler to be closed
         */
        @Override
        public void onActive(String topicPath, RegisteredHandler
registeredHandler) {
        }

        /**
         * @param topicPath
         *            - the branch of the topic tree for which
the handler was
         *            registered
         */
        @Override
        public void onClose(String topicPath) {
        }

        /**
         * @param notification
         *            - the missing topic details
         */
        @Override
        public void onMissingTopic(MissingTopicNotification
notification) {
            // Create a topic and do process() in the callback
            topicControl.addTopic(notification.getTopicPath(),
details, new AddTopicCallback(notification));
        }
    }

    private final class AddTopicCallback implements
TopicControl.AddCallback {
        private MissingTopicNotification notification;

        public AddTopicCallback(MissingTopicNotification
notification) {
            this.notification = notification;
        }

        @Override
        public void onDiscard() {
        }

        /**
         * @param topicPath
         *            - the topic path as supplied to the add
request
         * @param reason
         *            - the reason for failure
         */
        @Override
        public void onTopicAddFailed(String topicPath,
TopicAddFailReason reason) {
            // Cancel the notification because the server have
failed to
            notification.cancel();
        }

        /**
         * @param topicPath
         *            - the full path of the topic that was
added
         */
```

```
        @Override
        public void onTopicAdded(String topicPath) {
            // Proceed the notification
            notification.proceed();
        }

    }

}
```

.NET

```
using System.Threading.Tasks;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 public class ControlClientMissingTopicNotification
 {
  #region Fields

  private readonly ISession clientSession;
  private readonly ITopicControl topicControl;

  #endregion Fields

  #region Constructor

  public ControlClientMissingTopicNotification()
  {
   clientSession =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topicControl =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://
diffusion.example.com:80" ).GetTopicControlFeature();

   Subscribe( "some/path10" );
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Subscribes to a topic which may or may not be missing.
  /// </summary>
  /// <param name="topicPath">The path of the topic to subscribe
 to.</param>
  public async void Subscribe( string topicPath )
  {
   var missingTopicHandler = new MissingTopicHandler();

   // Add the 'missing topic handler' to the topic control
 object
```

```
  topicControl.AddMissingTopicHandler( topicPath,
missingTopicHandler );

  // Wait for the successful registration of the handler
  var registeredHandler = await
missingTopicHandler.OnActiveCalled;

  var topics = clientSession.GetTopicsFeature();

  var topicCompletion = new TaskCompletionSource<bool>();

  // Attempt to subscribe to the topic
  topics.Subscribe( topicPath, new
TopicsCompletionCallback( topicCompletion ) );

  await topicCompletion.Task;

  // Wait and see if a missing topic notification is generated
  var request = await missingTopicHandler.OnMissingTopicCalled;

  // Cancel the client request on the server
  request.Cancel();

  // Close the registered handler
  registeredHandler.Close();

  // All events in Diffusion are asynchronous, so we must wait
for the close to happen
  await missingTopicHandler.OnCloseCalled;
 }

 #endregion Public Methods

 #region Private Classes

 private class TopicsCompletionCallback :
ITopicsCompletionCallback
 {
  #region Fields

  private readonly TaskCompletionSource<bool>
theCompletionSource;

  #endregion Fields

  #region Constructor

  public TopicsCompletionCallback( TaskCompletionSource<bool>
source )
  {
   theCompletionSource = source;
  }

  #endregion Constructor

  /// <summary>
  /// This is called to notify that a call context was closed
prematurely, typically due to a timeout or the
  /// session being closed.  No further calls will be made for
the context.
  /// </summary>
  public void OnDiscard()
  {
```

```
    theCompletionSource.SetResult( false );
   }

   /// <summary>
   /// Called to indicate that the requested operation has been
processed by the server.
   /// </summary>
   public void OnComplete()
   {
    theCompletionSource.SetResult( true );
   }
  }

  /// <summary>
  /// Asynchronous helper class for handling missing topic
notifications.
  /// </summary>
  private class MissingTopicHandler : IMissingTopicHandler
  {
   private readonly TaskCompletionSource<IRegisteredHandler>
onActive =
    new TaskCompletionSource<IRegisteredHandler>();

   private readonly
TaskCompletionSource<IMissingTopicNotification> onMissingTopic
=
    new TaskCompletionSource<IMissingTopicNotification>();

   private readonly TaskCompletionSource<bool> onClose = new
TaskCompletionSource<bool>();

   /// <summary>
   /// Waits for the 'OnActive' event to be called.
   /// </summary>
   public Task<IRegisteredHandler> OnActiveCalled { get { return
onActive.Task; } }

   /// <summary>
   /// Waits for the 'OnMissingTopic' event to be called.
   /// </summary>
   public Task<IMissingTopicNotification> OnMissingTopicCalled
{ get { return onMissingTopic.Task; } }

   public Task OnCloseCalled { get { return onClose.Task; } }

   /// <summary>
   /// Called when a client session requests a topic that does
not exist, and the topic path belongs to part of
   /// the topic tree for which this handler was registered.
   ///
   /// The handler implementation should take the appropriate
action (for example, create the topic), and then call
   /// IMissingTopicNotification.Proceed on the supplied
notification.  This allows the client request to continue
   /// and successfully resolve against the topic if it was
created.
   ///
   /// A handler should always call Proceed() otherwise
resources will continue to be reserved on the server and the
   /// client's request will not complete.
   /// </summary>
   /// <param name="notification">The client notification
object.</param>
```

```
  void
IMissingTopicHandler.OnMissingTopic( IMissingTopicNotification
notification )
  {
   onMissingTopic.SetResult( notification );
  }

  /// <summary>
  /// Called when the handler has been successfully registered
with the server.
  ///
  /// A session can register a single handler of each type for
a given branch of the topic tree.  If there is
  /// already a handler registered for the topic path the
operation will fail, <c>registeredHandler</c> will be closed,
  /// and the session error handler will be notified.  To
change the handler, first close the previous handler.
  /// </summary>
  /// <param name="topicPath">The path that the handler is
active for.</param>
  /// <param name="registeredHandler">Allows the handler to be
closed.</param>
  void ITopicTreeHandler.OnActive( string topicPath,
IRegisteredHandler registeredHandler )
  {
   onActive.SetResult( registeredHandler );
  }

  /// <summary>
  /// Called if the handler is closed.  This happens if the
call to register the handler fails, or the handler
  /// is unregistered.
  /// </summary>
  /// <param name="topicPath">The branch of the topic tree for
which the handler was registered.</param>
  void ITopicTreeHandler.OnClose( string topicPath )
  {
   onClose.TrySetResult( false );
  }
 }

 #endregion Private Classes
}
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

TopicUpdateControl

Use the TopicUpdateControl feature to enable a client session to update topics.

The TopicUpdateControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
TopicUpdateControl	YES	YES	YES	PARTIAL Does not support paged topics

Related Links

Features on page 169

Units of functionality within the Unified API are exposed as features.

Updating topics
A client can use the TopicUpdateControl feature to update topics.

A client can update a topic in one of the following ways:

Exclusive updating

By registering with the Diffusion server as an update source for the branch of the topic tree that contains the topic to be updated.

If a client is registered as the active update source for a branch of the topic tree, no other clients can update topics in that branch of the topic tree.

Non-exclusive updating

By getting a non-exclusive updater from the TopicUpdateControl feature. This updater can be used to update any topic that does not aready have an active update source registered against it.

Registering as an update source

Required permissions: update_topic, register_handler

A client must register as an update source for a branch of the topic tree to be able to exclusively publish content to topics in that branch. This locks the branch of the topic tree and prevents other clients from publishing updates to topics in the branch.

When a client registers as an update source the following events occur:

1. The client requests to register as an update source on a branch of the topic tree.
   - The update source is registered against a topic path. This is a path that identifies a branch of the topic tree, for example foo/bar. The update source is registered as a source for that branch of the topic tree, for example the topics foo/bar/baz and foo/bar/fred/qux are included in the specified branch of the topic tree.
   - You cannot register an update source above or below an existing update source. For example, if a client has registered an update source against foo/bar/fred another client cannot register an update source against foo/bar or foo/bar/fred/qux.
   - You can register an update source against a topic owned by an existing publisher or a topic that has an update source created by the server that is used for topic failover.
2. The server validates the registration request and returns one of the following responses:
   - If the request is valid, the Diffusion server calls the `OnRegister` callback of the update source and passes a `RegisteredHandler` that you can use to deregister the update source.
   - If the request is not valid, the Diffusion server calls the `onClose` callback of the update source.

     For example, a registration request is not valid if it registers against a topic branch that is above or below a branch where an existing update source is registered.
3. When the update source is registered, the Diffusion server calls one of the following callbacks:
   - If the update source is the primary update source, the Diffusion server calls the `onActive` callback of the update source.
   - If another update source is already the primary source for this branch of the topic tree, the Diffusion server calls the `onStandby` callback of the update source.

4. If an update source is on standby, the update source cannot update the topics it is registered against. If the active update source for a branch of the topic tree closes or becomes inactive, a standby update source can then become active and become the primary update source for that branch of the topic tree.
5. If an update source is active, the Diffusion server provides the update source with an `Updater`. The update source can use the `Updater` to update the topics it is registered against.
6. If an active update source exists for a branch of the topic tree, no other clients can update topics in that branch of the topic tree.

Updating a topic non-exclusively

Required permissions: update_topic

To non-exclusively update topics, a client must get a non-exclusive updater from the TopicUpdateControl feature. This updater can be used to update any topic under the following conditions:

- The topic does not aready have an active update source registered against it
- The client has the update_topic permission for the topic

Using an updater to publish content to topics

Required permissions: update_topic

A client uses an updater to publish content to topics. Updaters can be used for exclusive or non-exclusive updating, depending on how the updater is acquired. When the client uses an updater method to publish content, it passes in the following parameters:

Topic path

The path to the topic to be updated.

If the updater is an exclusive updater, this topic must be in the branch of the topic tree that the client is the active update source for and that the updater is associated with.

Content

The information about the update can be provided as either a simple `Content` object or as a more complex `Update` object.

The content that is to be published to the topic. The client must use the appropriate content type when formatting the content. If the content uses the wrong content type for the topic, it can cause an error.

Update

The information about the update can be provided as either a simple `Content` object or as a more complex `Update` object.

An update that contains the content that is to be published to the topic and other information about the update, such as its type.

Use the `Builder` methods provided in the Unified API to build your `Update` objects.

Context

OPTIONAL: A context object can be passed in to the update method that provides application state information.

Callback

The server uses the callback to return the result of the update. If the update completes successfully, the Diffusion server calls the callback's `onComplete` method. Otherwise, the Diffusion server calls the callback's `onError` method.

Related Links

Failover of active update sources on page 87
You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

Building updates for paged topics

Use the `UpdateFactory` classes with the TopicUpdateControl feature to create the `Update` objects that can be used to update paged topics.

You can update a paged topic from the control client in the same way as you update any other topic: by using an updater to publish content to the topic. When updating a paged topic, you must pass an `Update` object to the updater. You can use update factories to create the `Update` object. The following update factory interfaces are provided for use with paged topics:

`PagedRecordOrderedUpdateFactory`

For use when the topic is a paged record topic and has an ordering policy of declared or comparator.

`PagedRecordUnorderedUpdateFactory`

For use when the topic is a paged record topic and has an ordering policy of unordered.

`PagedStringOrderedUpdateFactory`

For use when the topic is a paged string topic and has an ordering policy of declared or comparator.

`PagedStringUnorderedUpdateFactory`

For use when the topic is a paged string topic and has an ordering policy of unordered.

Ensure that you use the appropriate update factory for the type of paged topic you are updating.

Building updates for ordered topics

When building an update for an ordered topic, you can instruct the Diffusion server to add a line, remove a line, or update a line in the topic. The server uses the ordering policy and duplicates policy of the paged topic to determine where the line is added, whether the line is added, or which line is removed or updated.

Building updates for unordered topics

When building an update for an unordered topic, you can instruct the Diffusion server to add one or more lines, insert one or more lines at a specified index, update the line at a specified index, or remove one or more lines starting at a specified index.

Example: Make exclusive updates to a topic

The following examples use the Unified API to register as the update source of a topic and to update that topic with content. A client that updates a topic using this method locks the topic and prevents other clients from updating the topic.

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.SingleValueTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client as an event feed to a
 topic.
 * <P>
 * This uses the 'TopicControl' feature to create a topic and
 the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientAsUpdateSource {

    private static final String TOPIC_NAME = "Feeder";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final UpdateCallback updateCallback;

    /**
     * Constructor.
     *
     * @param callback for updates
     */
```

```
    public ControlClientAsUpdateSource(UpdateCallback callback)
{

        updateCallback = callback;

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");
        topicControl = session.feature(TopicControl.class);
        updateControl =
session.feature(TopicUpdateControl.class);
    }

    /**
     * Start the feed.
     *
     * @param provider the provider of prices
     * @param scheduler a scheduler service to schedule a
periodic feeder task
     */
    public void start(
        final PriceProvider provider,
        final ScheduledExecutorService scheduler) {

        // Set up topic details
        final SingleValueTopicDetails.Builder builder =
            topicControl

.newDetailsBuilder(SingleValueTopicDetails.Builder.class);
        final TopicDetails details =

builder.metadata(Diffusion.metadata().decimal("Price")).build();

        // Declare a custom update source implementation. When
the source is set
        // as active start a periodic task to poll the provider
every second and
        // update the topic. When the source is closed, stop the
scheduled task.
        final UpdateSource source = new UpdateSource.Default() {
            private ScheduledFuture<?> theFeeder;

            @Override
            public void onActive(String topicPath, Updater
updater) {
                theFeeder =
                    scheduler.scheduleAtFixedRate(
                        new FeederTask(provider, updater),
                        1, 1, TimeUnit.SECONDS);
            }

            @Override
            public void onClose(String topicPath) {
                if (theFeeder != null) {
                    theFeeder.cancel(true);
                }
            }
        };

        // Create the topic. When the callback indicates that
the topic has been
        // created then register the topic source for the topic.
```

```
        topicControl.addTopic(
            TOPIC_NAME,
            details,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    updateControl.registerUpdateSource(topic,
source);
                }
            });

    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">" + TOPIC_NAME,
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }

    /**
     * Periodic task to poll from provider and send update to
server.
     */
    private final class FeederTask implements Runnable {

        private final PriceProvider priceProvider;
        private final Updater priceUpdater;

        private FeederTask(PriceProvider provider, Updater
updater) {
            priceProvider = provider;
            priceUpdater = updater;
        }

        @Override
        public void run() {
            priceUpdater.update(
                TOPIC_NAME,

Diffusion.content().newContent(priceProvider.getPrice()),
                updateCallback);
        }

    }

    /**
     * Interface of a price provider that can periodically be
polled for a
```

```
     * price.
     */
    public interface PriceProvider {
        /**
         * Get the current price.
         *
         * @return current price as a decimal string
         */
        String getPrice();
    }
}
```

.NET

```
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client as an event feed to a
 topic.
 ///
 /// This uses the <see cref="ITopicControl"/> feature to create
 a topic and the <see cref="ITopicUpdateControl"/>
 /// feature to send updates to it.
 ///
 /// To send updates to a topic, the client session requires the
 <see cref="TopicPermission.UPDATE_TOPIC"/>
 /// permission for that branch of the topic tree.
 /// </summary>
 public class ControlClientAsUpdateSource
 {
  #region Fields

  private const string TopicName = "Feeder";

  private readonly ISession session;
  private readonly ITopicControl topicControl;
  private readonly ITopicUpdateControl updateControl;
  private readonly ITopicUpdaterUpdateCallback updateCallback;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  /// <param name="callback">The callback for updates.</param>
  public
 ControlClientAsUpdateSource( ITopicUpdaterUpdateCallback
 callback )
  {
   updateCallback = callback;
```

```
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com;80" );

  topicControl = session.GetTopicControlFeature();
  updateControl = session.GetTopicUpdateControlFeature();
 }

 #endregion Constructor

 #region Public Methods

 public void Start( IPriceProvider provider )
 {
  // Set up topic details
  var builder =
topicControl.CreateDetailsBuilder<ISingleValueTopicDetailsBuilder>();
  var details =
builder.Metadata( Diffusion.Metadata.Decimal( "Price" ) ).Build();

  // Declare a custom update source implementation.  When the
source is set as active, start a periodic task
  // to poll the provider every second and update the topic.
When the source is closed, stop the scheduled
  // task.
  var source = new UpdateSource( provider, updateCallback );

  // Create the topic.  When the callback indicates that the
topic has been created, register the topic
  // source for the topic
  topicControl.AddTopicFromValue( TopicName, details, new
AddCallback( updateControl, source ) );
 }

 public void Close()
 {
  // Remove our topic and close the session when done.
  topicControl.RemoveTopics( ">" + TopicName, new
RemoveCallback( session ));
 }

 #endregion Public Methods

 #region Private Classes

 private class RemoveCallback :
TopicControlRemoveCallbackDefault
 {
  #region Fields

  private readonly ISession theSession;

  #endregion Fields

  #region Constructor

  public RemoveCallback( ISession session )
  {
   theSession = session;
  }

  #endregion Constructor
```

```
  #region Overrides

  /// <summary>
  /// Notification that a call context was closed prematurely,
typically due to a timeout or the session being
  /// closed.  No further calls will be made for the context.
  /// </summary>
  public override void OnDiscard()
  {
   theSession.Close();
  }

  /// <summary>
  /// Topic(s) have been removed.
  /// </summary>
  public override void OnTopicsRemoved()
  {
   theSession.Close();
  }

  #endregion Overrides
 }

 private class AddCallback : TopicControlAddCallbackDefault
 {
  #region Fields

  private readonly ITopicUpdateControl updateControl;
  private readonly UpdateSource updateSource;

  #endregion Fields

  #region Constructor

  public AddCallback( ITopicUpdateControl updater, UpdateSource
source )
  {
   updateControl = updater;
   updateSource = source;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
  /// Topic has been added.
  /// </summary>
  /// <param name="topicPath">The full path of the topic that
was added.</param>
  public override void OnTopicAdded( string topicPath )
  {
   updateControl.RegisterUpdateSource( topicPath,
updateSource );
  }

  #endregion Overrides
 }

 private class UpdateSource : TopicUpdateSourceDefault
 {
  #region Fields
```

```
  private readonly IPriceProvider thePriceProvider;
  private readonly ITopicUpdaterUpdateCallback
theUpdateCallback;
  private readonly CancellationTokenSource cancellationToken =
new CancellationTokenSource();

  #endregion Fields

  #region Constructor

  public UpdateSource( IPriceProvider provider,
ITopicUpdaterUpdateCallback callback )
  {
   thePriceProvider = provider;
   theUpdateCallback = callback;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
  /// State notification that this source is now active for the
specified topic path, and is therefore in a valid
  /// state to send updates on topics at or below the
registered topic path.
  /// </summary>
  /// <param name="topicPath">The registration path.</param>
  /// <param name="updater">An updater that may be used to
update topics at or below the registered path.</param>
  public override void OnActive( string topicPath,
ITopicUpdater updater )
  {
   PeriodicTaskFactory.Start( () =>
   {
    updater.Update(
     TopicName,
Diffusion.Content.NewContent( thePriceProvider.Price ),
theUpdateCallback );
   }, 1000, cancelToken:cancellationToken.Token );
  }

  /// <summary>
  /// Called if the handler is closed. The handler will be
closed if the
  /// session is closed after the handler has been registered,
or if the
  /// handler is unregistered using <see
cref="IRegistration.Close">close</see>.
  ///
  /// No further calls will be made for the handler.
  /// </summary>
  /// <param name="topicPath">the branch of the topic tree for
which the handler was registered</param>
  public override void OnClose( string topicPath )
  {
   cancellationToken.Cancel();
  }

  #endregion Overrides
 }

 #endregion Private Classes
```

```
  #region Interfaces

  public interface IPriceProvider
  {
   /// <summary>
   /// Get the current price as a decimal string.
   /// </summary>
   string Price { get; }
  }

  #endregion Interfaces
 }
}
```

C

```
/*
 * This example creates a simple single-value topic and
 periodically updates
 * the data it contains.
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"
#include "service/svc-update.h"

int active = 0;

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'t', "topic", "Topic name to create and update",
 ARG_OPTIONAL, ARG_HAS_VALUE, "time"},
        END_OF_ARG_OPTS
};

/*
 * Handlers for add topic feature.
 */
static int
on_topic_added(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
```

```
        printf("Added topic\n");
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
        printf("Failed to add topic (%d)\n", response-
>response_code);
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

/*
 * Handlers for registration of update source feature
 */
static int
on_update_source_init(SESSION_T *session,
                      const CONVERSATION_ID_T *updater_id,
                      const SVC_UPDATE_REGISTRATION_RESPONSE_T
 *response,
                      void *context)
{
        printf("Topic source \"%s\" in init state\n",
 conversation_id_to_string(*updater_id));
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

static int
on_update_source_registered(SESSION_T *session,
                            const CONVERSATION_ID_T *updater_id,
                            const
 SVC_UPDATE_REGISTRATION_RESPONSE_T *response,
                            void *context)
{
        printf("Registered update source \"%s\"\n",
 conversation_id_to_string(*updater_id));
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

static int
on_update_source_active(SESSION_T *session,
                        const CONVERSATION_ID_T *updater_id,
                        const SVC_UPDATE_REGISTRATION_RESPONSE_T
 *response,
                        void *context)
{
        printf("Topic source \"%s\" active\n",
 conversation_id_to_string(*updater_id));
        active = 1;
        apr_thread_cond_broadcast(cond);
```

```
        return HANDLER_SUCCESS;
}

static int
on_update_source_standby(SESSION_T *session,
                         const CONVERSATION_ID_T *updater_id,
                         const
 SVC_UPDATE_REGISTRATION_RESPONSE_T *response,
                         void *context)
{
        printf("Topic source \"%s\" on standby\n",
 conversation_id_to_string(*updater_id));
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

static int
on_update_source_closed(SESSION_T *session,
                        const CONVERSATION_ID_T *updater_id,
                        const SVC_UPDATE_REGISTRATION_RESPONSE_T
 *response,
                        void *context)
{
        printf("Topic source \"%s\" closed\n",
 conversation_id_to_string(*updater_id));
        apr_thread_cond_broadcast(cond);
        return HANDLER_SUCCESS;
}

/*
 * Handlers for update of data.
 */
static int
on_update_success(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
        printf("on_update_success for updater \"%s\"\n",
 conversation_id_to_string(*updater_id));
        return HANDLER_SUCCESS;
}

static int
on_update_failure(SESSION_T *session,
                  const CONVERSATION_ID_T *updater_id,
                  const SVC_UPDATE_RESPONSE_T *response,
                  void *context)
{
        printf("on_update_failure for updater \"%s\"\n",
 conversation_id_to_string(*updater_id));
        return HANDLER_SUCCESS;
}

/*
 *
 */
int
main(int argc, char** argv)
{
        // Standard command line parsing.
        const HASH_T *options = parse_cmdline(argc, argv,
 arg_opts);
```

```
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        const char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }
        const char *topic_name = hash_get(options, "topic");

        // Setup for condition variable
        apr_initialize();
        apr_pool_create(&pool, NULL);
        apr_thread_mutex_create(&mutex,
 APR_THREAD_MUTEX_DEFAULT, pool);
        apr_thread_cond_create(&cond, pool);

        // Setup for session
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, principal, credentials,
 NULL, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        const TOPIC_DETAILS_T *string_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_STRING);
        const ADD_TOPIC_PARAMS_T add_topic_params = {
                .topic_path = topic_name,
                .details = string_topic_details,
                .on_topic_added = on_topic_added,
                .on_topic_add_failed = on_topic_add_failed,
                .on_discard = on_topic_add_discard,
        };

        apr_thread_mutex_lock(mutex);
        add_topic(session, add_topic_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        // Handlers for add_update_source()
        const UPDATE_SOURCE_REGISTRATION_PARAMS_T
 update_reg_params = {
                .topic_path = topic_name,
                .on_init = on_update_source_init,
                .on_registered = on_update_source_registered,
                .on_active = on_update_source_active,
                .on_standby = on_update_source_standby,
                .on_close = on_update_source_closed
        };

        // Register an updater
        apr_thread_mutex_lock(mutex);
```

```
        const CONVERSATION_ID_T *updater_id =
 register_update_source(session, update_reg_params);
        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        UPDATE_SOURCE_PARAMS_T update_source_params_base = {
                .updater_id = updater_id,
                .topic_path = topic_name,
                .on_success = on_update_success,
                .on_failure = on_update_failure
        };

        while(active) {
                BUF_T *buf = buf_create();
                const time_t time_now = time(NULL);
                buf_write_string(buf, ctime(&time_now));

                CONTENT_T *content =
 content_create(CONTENT_ENCODING_NONE, buf);

                UPDATE_T *upd =
 update_create(UPDATE_ACTION_REFRESH,

 UPDATE_TYPE_CONTENT,
                                                content);

                UPDATE_SOURCE_PARAMS_T update_source_params =
 update_source_params_base;
                update_source_params.update = upd;

                update(session, update_source_params);

                content_free(content);

                sleep(1);
        }

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Make non-exclusive updates to a topic

The following examples use the Unified API to update a topic with content. Updating a topic this way does not prevent other clients from updating the topic.

JavaScript

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. A session may update any existing topic. Update values
 must be of the same type as the topic being updated.

    // Add a topic first with a string type
    session.topics.add('foo', '').then(function() {
        // Update the topic
        return session.topics.update('foo', 'hello');
    }).then(function() {
```

```
        // Update the topic again
        return session.topics.update('foo', 'world');
    });

    // 2. If using RecordContent metadata, update values are
 constructed from the metadata

    // Create a new metadata instance
    var meta = new diffusion.metadata.RecordContent();

    meta.addRecord('record', 1, {
        'field' : meta.integer()
    });

    // Create a builder to set values
    var builder = meta.builder();

    builder.add('record', {
        field : 123
    });

    // Update the topic with the new value
    session.topics.add('topic', '').then(function() {
        session.topics.update('topic', builder.build());
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update a
 topic in non
 * exclusive mode (as opposed to acting as an exclusive update
 source). In this
 * mode other clients could update the same topic (on a last
 update wins basis).
 * <P>
 * This uses the 'TopicControl' feature to create a topic and
 the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * To send updates to a topic, the client session requires the
 'update_topic'
 * permission for that branch of the topic tree.
 *
```

```
 * @author Push Technology Limited
 * @since 5.3
 */
public final class ControlClientUpdatingTopic {

    private static final String TOPIC = "MyTopic";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;

    /**
     * Constructor.
     */
    public ControlClientUpdatingTopic() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl =
 session.feature(TopicUpdateControl.class);

        // Create a single value topic
        topicControl.addTopic(
            TOPIC,
            TopicType.SINGLE_VALUE,
            new AddCallback.Default());

    }

    /**
     * Update the topic with a string value.
     *
     * @param value the update value
     * @param callback the update callback
     */
    public void update(String value, UpdateCallback callback) {
        updateControl.updater().update(TOPIC, value, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">" + TOPIC,
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
```

```
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;

namespace UCIStack.Examples
{
    /// <summary>
    /// An example of using a control client to create and
 update a topic in non-exclusive mode (as opposed to acting
    /// as an exclusive update source).  In this mode other
 clients could update the same topic (on a 'last update wins'
    /// basis).
    ///
    /// This uses the <see cref="ITopicControl"/> feature
 to create a topic and the <see cref="ITopicUpdateControl"/>
 feature
    /// to send updates to it.
    ///
    /// To send updates to a topic, the client session requires
 the 'update_topic' permission for that branch of the
    /// topic tree.
    /// </summary>
    public class ControlClientUpdatingTopic
    {
        #region Fields

        private const string Topic = "MyTopic";

        private readonly ISession session;
        private readonly ITopicControl topicControl;
        private readonly ITopicUpdateControl updateControl;

        #endregion Fields

        #region Constructor

        /// <summary>
        /// Constructor.
        /// </summary>
        public ControlClientUpdatingTopic()
        {
            session = Diffusion.Sessions.Principal( "control" )
                .Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            topicControl = session.GetTopicControlFeature();
            updateControl =
 session.GetTopicUpdateControlFeature();

            // Create a single-value topic.
            topicControl.AddTopicFromValue( Topic,
 TopicType.SINGLE_VALUE, new TopicControlAddCallbackDefault() );
        }

        #endregion Constructor
```

```
        #region Public Methods

        /// <summary>
        /// Update the topic with a string value.
        /// </summary>
        /// <param name="value">The update value.</param>
        /// <param name="callback">The update callback.</param>
        public void Update( string value,
ITopicUpdaterUpdateCallback callback )
        {
            updateControl.Updater.Update( Topic, value,
callback );
        }

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close()
        {
            // Remove our topic and close session when done.
            topicControl.RemoveTopics( ">" + Topic, new
RemoveCallback( session ) );
        }

        #endregion Public Methods

        #region Private Classes

        private class RemoveCallback :
TopicControlRemoveCallbackDefault
        {
            #region Fields

            private readonly ISession theSession;

            #endregion Fields

            #region Constructor

            public RemoveCallback( ISession session )
            {
                theSession = session;
            }

            #endregion Constructor

            #region Overrides

            /// <summary>
            /// Notification that a call context was closed
prematurely, typically due to a timeout or the session being
            /// closed.  No further calls will be made for the
context.
            /// </summary>
            public override void OnDiscard()
            {
                theSession.Close();
            }

            /// <summary>
            /// Topic(s) have been removed.
            /// </summary>
            public override void OnTopicsRemoved()
```

```
            {
                theSession.Close();
            }

            #endregion Overrides
        }

        #endregion Private Classes
    }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Update a paged topic
The following examples use the Unified API to create a paged topic and to update that paged topic with content.

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.Collection;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.content.update.PagedRecordOrderedUpdat
import
 com.pushtechnology.diffusion.client.content.update.PagedStringUnorderedUpd
import
 com.pushtechnology.diffusion.client.content.update.Update;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update
 paged topics.
 * <P>
 * This uses the 'TopicControl' feature to create a paged topic
 and the
 * 'TopicUpdateControl' feature to send updates to it.
```

```
 * <P>
 * This demonstrates some simple examples of paged topic updates
 but not all of
 * the possible ways in which they can be done.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientUpdatingPagedTopics {

    private static final String ORDERED_TOPIC = "Paged/Ordered";
    private static final String UNORDERED_TOPIC = "Paged/
Unordered";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final PagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
    private final PagedStringUnorderedUpdateFactory
 unorderedUpdateFactory;
    private Updater pagedUpdater = null;

    /**
     * Constructor.
     */
    public ControlClientUpdatingPagedTopics() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl =
 session.feature(TopicUpdateControl.class);

        orderedUpdateFactory =
            updateControl

 .updateFactory(PagedRecordOrderedUpdateFactory.class);
        unorderedUpdateFactory =
            updateControl

 .updateFactory(PagedStringUnorderedUpdateFactory.class);

        final MetadataFactory metadata = Diffusion.metadata();

        // Create an unordered paged string topic
        topicControl.addTopic(
            UNORDERED_TOPIC,
            topicControl.newDetails(TopicType.PAGED_STRING),
            new AddCallback.Default());

        // Create an ordered paged record topic
        final MRecord recordMetadata =
            metadata.record(
                "Record",
                metadata.string("Name"),
                metadata.string("Address"));
```

```
        topicControl.addTopic(
            ORDERED_TOPIC,
            topicControl.

newDetailsBuilder(PagedRecordTopicDetails.Builder.class).
                metadata(recordMetadata).
                order(new OrderKey("Name")).build(),
            new AddCallback.Default());

        // Register as updater for topics under the 'Paged'
branch
        updateControl.registerUpdateSource(
            "Paged",
            new UpdateSource.Default() {
                @Override
                public void onActive(String topicPath, Updater
updater) {
                    pagedUpdater = updater;
                }
            });
    }

    /**
     * Add a new line to the ordered topic.
     *
     * @param name the name field value
     * @param address the address field value
     * @param callback to notify result
     */
    public void addOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.add(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Update a line of an ordered topic.
     *
     * @param name the name of the line to update
     * @param address the new address field value
     * @param callback to notify result
     */
    public void updateOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.update(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Remove a line from an ordered topic.
```

```
     *
     * @param name the name of the line to remove
     * @param callback to notify result
     */
    public void removeOrdered(String name, UpdateCallback
callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.remove(
                Diffusion.content().newRecord(name, "")),
            callback);
    }

    /**
     * Add a line or lines to the end of an unordered topic.
     *
     * @param values lines to add
     * @param callback to notify result
     */
    public void addUnordered(
        Collection<String> values,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.add(values),
            callback);
    }

    /**
     * Insert a line or lines at a specified index within an
unordered topic.
     *
     * @param index the index to add at
     * @param values lines to insert
     * @param callback to notify result
     */
    public void insertUnordered(
        int index,
        Collection<String> values,
        UpdateCallback callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.insert(index, values),
            callback);
    }

    /**
     * Update a line within an unordered topic.
     *
     * @param index the index of the line to update
     * @param value the new line value
     * @param callback to notify result
     */
    public void updateUnordered(
        int index,
        String value,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.update(index, value),
```

```
            callback);
    }

    /**
     * Remove a specific line from an unordered topic.
     *
     * @param index the line to remove
     * @param callback to notify result
     */
    public void removeUnordered(int index, UpdateCallback
 callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.remove(index),
            callback);
    }

    private void update(String topic, Update update,
 UpdateCallback callback)
        throws IllegalStateException {
        if (pagedUpdater == null) {
            throw new IllegalStateException("No updater");
        }
        pagedUpdater.update(topic, update, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">Paged",
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
}
```

.NET

```
using System;
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Topics.Update;

namespace UCIStack.Examples
{
 /// <summary>
```

```
/// An example of using a control client to create and update
paged topics.
///
/// This uses the <see cref="ITopicControl"/> feature to create
a paged topic and the <see cref="ITopicUpdateControl"/>
/// feature to send updates to it.
///
/// This demonstrates some simple examples of paged topic
updates but not all of the possible ways in which they
/// can be done.
///
/// To send updates to a topic, the client session requires the
UPDATE_TOPIC permission for that branch of the
/// topic tree.
/// </summary>
public class ControlClientUpdatingPagedTopics
{
 #region Fields

 private const string OrderedTopic = "Paged/Ordered";
 private const string UnorderedTopic = "Paged/Unordered";

 private readonly ISession session;
 private readonly ITopicControl topicControl;
 private readonly IPagedRecordOrderedUpdateFactory
orderedUpdateFactory;
 private readonly IPagedStringUnorderedUpdateFactory
unorderedUpdateFactory;
 private static ITopicUpdater _pagedUpdater;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 public ControlClientUpdatingPagedTopics()
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  topicControl = session.GetTopicControlFeature();
  var updateControl = session.GetTopicUpdateControlFeature();

  orderedUpdateFactory =
updateControl.UpdateFactory<IPagedRecordOrderedUpdateFactory>();
  unorderedUpdateFactory =
updateControl.UpdateFactory<IPagedStringUnorderedUpdateFactory>();

  var metadata = Diffusion.Metadata;

  // Create an unordered paged string topic
  topicControl.AddTopic(
   UnorderedTopic,
   topicControl.NewDetails( TopicType.PAGED_STRING ),
   new TopicControlAddCallbackDefault() );

  // Create an ordered paged record topic
  var recordMetadata = metadata.Record(
   "Record",
   metadata.String( "Name" ),
```

```
    metadata.String( "Address" ) );

   topicControl.AddTopic(
    OrderedTopic,
    topicControl
     .CreateDetailsBuilder<IPagedRecordTopicDetailsBuilder>()
     .Metadata( recordMetadata )
     .Order( new PagedRecordOrderKey( "Name" ) ).Build(),
    new TopicControlAddCallbackDefault() );

   // Register an updater for topics under the 'Paged' branch
   updateControl.RegisterUpdateSource( "Paged", new
UpdateSource() );
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Add a new line from an ordered topic.
  /// </summary>
  /// <param name="name">The name field value.</param>
  /// <param name="address">The address field value.</param>
  /// <param name="callback">The callback to notify the
result.</param>
  public void AddOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,
    orderedUpdateFactory.Add( Diffusion.Content.NewRecord( name,
address ) ),
    callback );
  }

  /// <summary>
  /// Update a line of an ordered topic.
  /// </summary>
  /// <param name="name">The name of the line to update.</param>
  /// <param name="address">The new address field value.</param>
  /// <param name="callback">The callback to notify the
result.</param>
  public void UpdateOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,

orderedUpdateFactory.Update( Diffusion.Content.NewRecord( name,
address ) ),
    callback );
  }

  /// <summary>
  /// Remove a line from an ordered topic.
  /// </summary>
  /// <param name="name">The name of the line to remove.</param>
  /// <param name="callback">The callback to notify the
result.</param>
  public void RemoveOrdered( string name,
ITopicUpdaterUpdateCallback callback )
  {
```

```
    Update(
     OrderedTopic,

 orderedUpdateFactory.Remove( Diffusion.Content.NewRecord( name,
 "" ) ),
     callback );
  }

  /// <summary>
  /// Add a line or lines to the end of an unordered topic.
  /// </summary>
  /// <param name="values">The lines to add.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void AddUnordered( ICollection<string> values,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Add( values ),
    callback );
  }

  /// <summary>
  /// Insert a line or lines at a specified index within an
 unordered topic.
  /// </summary>
  /// <param name="index">The index at which to add the line.</
param>
  /// <param name="values">The lines to insert.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void InsertUnordered( int index, ICollection<string>
 values, ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Insert( index, values ),
    callback );
  }

  /// <summary>
  /// Remove a specific line from an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to remove.</
param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveUnordered( int index,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Remove( index ),
    callback );

  }

  /// <summary>
  /// Update a line within an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to update.</
param>
```

```
 /// <param name="value">The new line value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void UpdateUnordered( int index, string value,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
   unorderedUpdateFactory.Update( index, value ),
   callback );
 }

 /// <summary>
 /// Close the session.
 /// </summary>
 public void Close()
 {
  // Remove our topics and close the session when done.
  topicControl.RemoveTopics( ">Paged", new
RemoveCallback( session ) );
 }

 #endregion Public Methods

 #region Private Methods

 private static void Update( string topic, IUpdate update,
ITopicUpdaterUpdateCallback callback )
 {
  if( _pagedUpdater == null )
  {
   throw new InvalidOperationException( "The paged updater has
not been initialised." );
  }

  _pagedUpdater.Update( topic, update, callback );
 }

 #endregion Private Methods

 #region Private Classes

 private class RemoveCallback :
TopicControlRemoveCallbackDefault
 {
  #region Fields

  private readonly ISession theSession;

  #endregion Fields

  #region Constructor

  public RemoveCallback( ISession session )
  {
   theSession = session;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
```

```
    /// Notification that a call context was closed prematurely,
  typically due to a timeout or the session being
    /// closed.  No further calls will be made for the context.
    /// </summary>
    public override void OnDiscard()
    {
     theSession.Close();
    }

    /// <summary>
    /// Topic(s) have been removed.
    /// </summary>
    public override void OnTopicsRemoved()
    {
     theSession.Close();
    }

    #endregion Overrides
   }

   private class UpdateSource : TopicUpdateSourceDefault
   {
    #region Overrides

    /// <summary>
    /// State notification that this source is now active for the
  specified topic path, and is therefore in a valid
    /// state to send updates on topics at or below the
  registered topic path.
    /// </summary>
    /// <param name="topicPath">The registration path.</param>
    /// <param name="updater">An updater that may be used to
  update topics at or below the registered path.</param>
    public override void OnActive( string topicPath,
  ITopicUpdater updater )
    {
     _pagedUpdater = updater;
    }

    #endregion Overrides
   }

   #endregion Private Classes
  }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

MessagingControl

Use the MessagingControl feature to enable a client to send messages to specific client sessions and receive messages sent by clients to topic paths.

The MessagingControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
MessagingControl	YES	YES	YES	PARTIAL Does not support sending messages to a specific client.

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Messaging to clients
A client can use the MessagingControl feature to send individual messages to any known client on any topic path. It can also register a handler for messages sent from clients.

Sending messages to clients

Required permissions: view_session, send_to_session

A client can send a message to any known client session on any topic path, regardless of whether a topic is bound to that topic path. The messages are delivered to other clients through the Messaging feature or through the topic listener mechanism.

The client requires the session identifier of the client session it is sending to. Use the ClientControl feature to get the identifiers for connected client sessions. A client can also send messages back to clients from which it receives messages through a *message handler*.

The body of the message that is sent is represented as *content*. Any of the content builder features can be used to build message content. With messaging you can also send empty content.

When sending a message certain additional options can also be specified:

Headers

A set of string values that can be sent along with the content.

Priority

Use this to specify the priority used when queuing the message for the client at the the Diffusion server.

Filtering message recipients using session properties

Required permissions: view_session

When sending a message, a client can specify a filter for the recipients of that message. The filter is a query expression on the values of session properties.

The client defines a filter and sends a message on a topic path with the filter associated. The Diffusion server evaluates the query and sends the message on to connected client sessions whose session properties match the filter.

Registering message handlers

Required permissions: register_handler

A client can add a message handler for any branch of the topic tree. This handler receives messages sent from clients on any topic in that branch unless overridden by a handler registered for a more specific branch.

Each client session can register only a single handler for any branch in the topic tree. To change the handler for a particular branch, the previous handler must be closed.

A message received by a message handler comprises content and message context which can include headers. The content can be empty. The client interprets the content of messages. If a topic is bound to the topic path that the message is sent on, message content is not required to match the content definition of that topic.

Requesting session properties with messages

Required permissions: register_handler,view_session

When registering a message handler, a client can specify session properties that it is interested in receiving with the handled messages.

Messages received by this message handler include the current values of the requested session properties for the client session that sent the message.

The message handler can request both fixed properties and user-defined properties of the session.

Related Links

Session properties on page 165
A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

Session filtering on page 166
Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

Example: Handle messages and send messages to sessions
The following examples use the MessagingControl feature in the Unified API to handle messages sent to topic paths and to send messages to one or more clients.

JavaScript

```
var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
 environment.
// Node.js will not accept self-signed certificates by default.
 If you have
// one of these, set the environment variable
 NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. Messages can be sent & received between sessions.

    // Create a stream of received messages for a specific path
    session.messages.listen('foo').on('message', function(msg) {
        console.log('Received message: ' + msg.content);
    });

    // Send a message to another session. It is the
 application's responsibility to find the SessionID of the
 intended
    // recipient.
    session.messages.send('foo', 'Hello world', 'another-
session-id');

    // 2. Messages can also be sent without a recipient, in
 which case they will be dispatched to any Message Handlers
    // that have been registered for the same path. If multiple
 handlers are registered to the same path, any given
```

```
    // message will only be dispatched to one handler.

    // Register the handler to receive messages at or below the
 given path.
    session.messages.addHandler('foo', {
        onActive : function() {
            console.log('Handler registered');
        },
        onClose : function() {
            console.log('Handler closed');
        },
        onMessage : function(msg) {
            console.log('Received message:' + msg.content + '
 from Session: ' + msg.session);
            if (msg.properties) {
                console.log('with properties:', msg.properties);
            }
        }
    }).then(function() {
        console.log('Registered handler');
    }, function(e) {
        console.log('Failed to register handler: ', e);
    });

    // Send a message at a lower path, without an explicit
 recipient - this will be received by the Handler.
    session.messages.send('foo/bar', 'Another message');
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;
import com.pushtechnology.diffusion.client.types.ReceiveContext;

/**
 * This is an example of a control client using the
 'MessagingControl' feature
 * to receive messages from clients and also send messages to
 clients.
 * <P>
 * It is a trivial example that simply responds to all messages
 on a particular
 * branch of the topic tree by echoing them back to the client
 exactly as they
 * are complete with headers.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientReceivingMessages {
```

```
    private final Session session;
    private final MessagingControl messagingControl;
    private final SendCallback sendCallback;

    /**
     * Constructor.
     *
     * @param callback for result of sends
     */
    public ControlClientReceivingMessages(SendCallback callback)
{

        sendCallback = callback;

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");
        messagingControl =
session.feature(MessagingControl.class);

        // Register to receive all messages sent by clients on
the "foo" branch
        // To do this, the client session must have the
'register_handler' permission.
        messagingControl.addMessageHandler("foo", new
EchoHandler());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

    /**
     * Handler that echoes messages back to the originating
client complete with
     * original headers.
     */
    private class EchoHandler extends MessageHandler.Default {
        @Override
        public void onMessage(
            SessionId sessionId,
            String topicPath,
            Content content,
            ReceiveContext context) {

            // To send a message to a client, this client
session must have
            // the 'view_session' and 'send_to_session'
permissions.
            messagingControl.send(
                sessionId,
                topicPath,
                content,
                messagingControl.sendOptionsBuilder()
                    .headers(context.getHeaderList())
                    .build(),
                sendCallback);

        }
```

```
    }

}
```

.NET

```
using System.Linq;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is an example of a control client using the <see
 cref="IMessagingControl"/> feature to receive messages
 /// from clients and also send messages to clients.
 ///
 /// It is a trivial example that simply responds to all
 messages on a particular branch of the topic tree by
 /// echoing them back to the client exactly as they are,
 complete with headers.
 /// </summary>
 public class ControlClientReceivingMessages
 {
  #region Fields

  private readonly ISession session;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  /// <param name="callback">The callback to receive the result
 of message sending.</param>
  public ControlClientReceivingMessages( ISendCallback
 callback )
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   var messagingControl = session.GetMessagingControlFeature();

   // Register to receive all messages sent by clients on the
 "foo" branch.
   // To do this, the client session must have the
 REGISTER_HANDLER permission.
   messagingControl.AddMessageHandler( "foo", new
 EchoHandler( messagingControl, callback ) );
  }

  #endregion Constructor

  #region Public Methods
```

```
  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods

  #region Private Classes

  private class EchoHandler : MessageHandlerDefault
  {
   #region Fields

   private readonly IMessagingControl theMessagingControl;
   private readonly ISendCallback theSendCallback;

   #endregion Fields

   #region Constructor

   public EchoHandler( IMessagingControl messagingControl,
 ISendCallback sendCallback )
   {
    theMessagingControl = messagingControl;
    theSendCallback = sendCallback;
   }

   #endregion Constructor

   #region Overrides

   /// <summary>
   /// Receives content sent from a session via a topic.
   /// </summary>
   /// <param name="sessionId">Identifies the client session
 that sent the content.</param>
   /// <param name="topicPath">The path of the topic that the
 content was sent on.</param>
   /// <param name="content">The content sent by the client.</
param>
   /// <param name="context">The context associated with the
 content.</param>
   public override void OnMessage(
    SessionId sessionId,
    string topicPath,
    IContent content,
    IReceiveContext context )
   {
    theMessagingControl.Send(
     sessionId,
     topicPath,
     content,
     theMessagingControl.CreateSendOptionsBuilder()
      .SetHeaders( context.HeadersList.ToList() )
      .Build(),
     theSendCallback );
   }

   #endregion Overrides
```

```
  }

  private class MessageHandlerDefault : TopicTreeHandlerDefault,
 IMessageHandler
  {
   /// <summary>
   /// Receives content sent from a session via a topic.
   /// </summary>
   /// <param name="sessionId">Identifies the client session
 that sent the content.</param>
   /// <param name="topicPath">The path of the topic that the
 content was sent on.</param>
   /// <param name="content">The content sent by the client.</
param>
   /// <param name="context">The context associated with the
 content.</param>
   public virtual void OnMessage( SessionId sessionId, string
 topicPath, IContent content, IReceiveContext context )
   {
   }
  }

  #endregion Private Classes
 }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Use session property filters with messaging
The following examples use the MessagingControl feature in the Unified API to request session properties with messages sent to topic paths and to send messages to one or more clients depending on the values of their session properties.

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;
import com.pushtechnology.diffusion.client.types.ReceiveContext;

/**
 * This is an example of a control client using the
 'MessagingControl' feature
 * to send messages to clients using message filters. It also
 demonstrates the
 * ability to register a message handler with an interest in
 session property
 * values.
 *
 * @author Push Technology Limited
 * @since 5.5
 */
public final class ControlClientUsingFiltersAndProperties {
```

```
   private final Session session;
   private final MessagingControl messagingControl;
   private final SendToFilterCallback sendToFilterCallback;

   /**
    * Constructor.
    *
    * @param callback for result of sends
    */
   public
ControlClientUsingFiltersAndProperties(SendToFilterCallback
callback) {

       sendToFilterCallback = callback;

       session =

Diffusion.sessions().principal("control").password("password")
               .open("ws://diffusion.example.com:80");
       messagingControl =
session.feature(MessagingControl.class);

       // Register to receive all messages sent by clients on
the "foo" branch
       // and include the "JobTitle" session property value
with each message.
       // To do this, the client session must have the
'register_handler'
       // permission.
       messagingControl.addMessageHandler(
           "foo", new BroadcastHandler(), "JobTitle");
   }

   /**
    * Close the session.
    */
   public void close() {
       session.close();
   }

   /**
    * Handler that will pass any message to all sessions that
have a "JobTitle"
    * property set to "Staff" if, and only if it comes from a
session that has
    * a "JobTitle" set to "Manager".
    */
   private class BroadcastHandler extends
MessageHandler.Default {
       @Override
       public void onMessage(
           SessionId sessionId,
           String topicPath,
           Content content,
           ReceiveContext context) {

           if
("Manager".equals(context.getSessionProperties().get("JobTitle")))
{
               messagingControl.sendToFilter(
                   "JobTitle is 'Staff'",
                   topicPath,
                   content,
```

```
                    messagingControl.sendOptionsBuilder()
                        .headers(context.getHeaderList())
                        .build(),
                    sendToFilterCallback);
            }

        }
    }

}
```

.NET

```
using System.Linq;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is an example of a control client using the
 'MessagingControl' feature to send messages to clients using
 /// message filters.  It also demonstrates the ability to
 register a message handler with an interest in session
 /// property values.
 /// </summary>
 public class ControlClientUsingFiltersAndProperties
 {
  #region Fields

  private readonly ISession theSession;
  private readonly IMessagingControl theMessagingControl;
  private readonly ISendToFilterCallback
 theSendToFilterCallback;

  #endregion Fields

  #region Constructor

  public
 ControlClientUsingFiltersAndProperties( ISendToFilterCallback
 callback )
  {
   theSendToFilterCallback = callback;

   theSession =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   theMessagingControl =
 theSession.GetMessagingControlFeature();

   // Register and receive all messages sent by clients on the
 "foo" branch and include the "JobTitle" session
   // property value with each message.  To do this, the client
 session must have the "register_handler"
   // permission.
```

```
   theMessagingControl.AddMessageHandler(
    "foo",
    new BroadcastHandler( theMessagingControl,
theSendToFilterCallback ),
    "JobTitle" );
 }

 #endregion Constructor

 #region Public Methods

 public void Close()
 {
  theSession.Close();
 }

 #endregion Public Methods

 #region Private Classes

 private class BroadcastHandler : IMessageHandler
 {
  #region Fields

  private readonly IMessagingControl theMessagingControl;
  private readonly ISendToFilterCallback
theSendToFilterCallback;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  /// <param name="messagingControl">The messaging control
object.</param>
  /// <param name="callback">The filter callback.</param>
  public BroadcastHandler( IMessagingControl messagingControl,
ISendToFilterCallback callback )
  {
   theMessagingControl = messagingControl;
   theSendToFilterCallback = callback;
  }

  #endregion Constructor

  /// <summary>
  /// Called when the handler has been successfully registered
with the server.
  ///
  /// A session can register a single handler of each type for
a given branch of the topic tree.  If there is
  /// already a handler registered for the topic path the
operation will fail, <c>registeredHandler</c> will be closed,
  /// and the session error handler will be notified.  To
change the handler, first close the previous handler.
  /// </summary>
  /// <param name="topicPath">The path that the handler is
active for.</param>
  /// <param name="registeredHandler">Allows the handler to be
closed.</param>
```

```
    public void OnActive( string topicPath, IRegisteredHandler
 registeredHandler )
    {
    }

    /// <summary>
    /// Called if the handler is closed.  This happens if the
 call to register the handler fails, or the handler
    /// is unregistered.
    /// </summary>
    /// <param name="topicPath">The branch of the topic tree for
 which the handler was registered.</param>
    public void OnClose( string topicPath )
    {
    }

    /// <summary>
    /// Receives content sent from a session via a topic.
    /// </summary>
    /// <param name="sessionId">Identifies the client session
 that sent the content.</param>
    /// <param name="topicPath">The path of the topic that the
 content was sent on.</param>
    /// <param name="content">The content sent by the client.</
param>
    /// <param name="context">The context associated with the
 content.</param>
    public void OnMessage( SessionId sessionId, string topicPath,
 IContent content, IReceiveContext context )
    {

 if( !"Manager".Equals( context.SessionProperties["JobTitle"] ) )
 return;

     theMessagingControl.SendToFilter(
      "JobTitle is 'Staff'",
      topicPath,
      content,

 theMessagingControl.CreateSendOptionsBuilder().SetHeaders( context.Headers
      theSendToFilterCallback );
    }
   }

   #endregion Private Classes
  }
 }
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Related Links

Session properties on page 165
A client session has a number of properties associated with it. Properties are key-value pairs. Both the key and the value are case sensitive.

Session filtering on page 166

Session filters enable you to query the set of connected client sessions on the Diffusion server based on their session properties.

AuthenticationControl

Use the AuthenticationControl feature to enable a client session to authenticate other clients.

The AuthenticationControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
AuthenticationControl	NO	YES	YES	YES

Related Links

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Authentication on page 142
You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Developing a control authentication handler on page 307
Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

Developing a composite control authentication handler on page 310
Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Features on page 169
Units of functionality within the Unified API are exposed as features.

Authenticating clients
A client can use the AuthenticationControl feature to authenticate other client sessions.

Registering a control authentication handler

Required permissions: authenticate, register_handler

A client can register an authentication handler that can be called when a client connects to the Diffusion server or changes the principal and credentials it is connected with.

The authentication handler can decide whether a client's authentication request is allowed or denied, or the authentication handler can abstain from the decision. In which case the next configured authentication handler is called.

If the authentication handler allows a client's authentication request, it can assign roles to that client's session.

For more information about authentication and role-based security, see Authentication on page 142.

Example: Register an authentication handler

The following examples use the Unified API to register a control authentication handler with the Diffusion server. The examples also include a simple or empty authentication handler.

The name by which the control authentication handler is registered must be configured in the `Server.xml` configuration file of the Diffusion server for the control authentication handler to be called to handle authentication requests.

Java

```
package com.pushtechnology.diffusion.examples;

import java.nio.charset.Charset;
import java.util.Arrays;
import java.util.EnumSet;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.details.SessionDetails;
import
 com.pushtechnology.diffusion.client.details.SessionDetails.DetailType;
import
 com.pushtechnology.diffusion.client.features.ServerHandler;
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticatio
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticatio
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.types.Credentials;

/**
 * This demonstrates the use of a control client to authenticate
 client
 * connections.
 * <P>
 * This uses the 'AuthenticationControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientIdentityChecks {

    private final Session session;

    /**
     * Constructor.
     */
    public ControlClientIdentityChecks() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        final AuthenticationControl authenticationControl =
            session.feature(AuthenticationControl.class);

        // To register the authentication handler, this client
 session must
        // have the 'authenticate' and 'register_handler'
 permissions.
        authenticationControl.setAuthenticationHandler(
            "example-handler",
```

```
            EnumSet.allOf(DetailType.class),
            new Handler());
    }

    /**
     * Authentication handler.
     */
    private static class Handler extends ServerHandler.Default
        implements ControlAuthenticationHandler {
        @Override
        public void authenticate(
            final String principal,
            final Credentials credentials,
            final SessionDetails sessionDetails,
            final Callback callback) {

            final byte[] passwordBytes =
                "password".getBytes(Charset.forName("UTF-8"));

            if ("admin".equals(principal) &&
                credentials.getType() ==
 Credentials.Type.PLAIN_PASSWORD &&
                Arrays.equals(credentials.toBytes(),
 passwordBytes)) {
                callback.allow();
            }
            else {
                callback.deny();
            }
        }
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}
```

.NET

```
using System;
using System.Linq;
using System.Threading;
using PushTechnology.ClientInterface.Client.Details;
using PushTechnology.ClientInterface.Client.Factories;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a control client which registers an authentication
 handler with a server.
 /// </summary>
 public class ControlAuthenticationClient
 {
  /// <summary>
  /// Main entry point.
  /// </summary>
  /// <param name="args">The command line arguments, currently
 ignored.</param>
  public static void Main( string[] args )
```

```
  {
   var session = Diffusion.Sessions
    .Principal( "auth" )
    .Password( "auth_secret" )
    .Open( "ws://diffusion.example.com:80" );


 session.GetAuthenticationControlFeature().SetAuthenticationHandler(
    "control-client-auth-handler-example",

 Enum.GetValues( typeof( DetailType ) ).OfType<DetailType>().ToList(),
    new ExampleControlAuthenticationHandler() );

   while( true )
   {
    Thread.Sleep( 60000 );
   }
   // ReSharper disable once FunctionNeverReturns
  }
 }
}
```

C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"

struct user_credentials_s {
        const char *username;
        const char *password;
};

static const struct user_credentials_s USERS[] = {
        { "fish", "chips" },
        { "ham", "eggs" },
        { NULL, NULL }
};

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'n', "name", "Name under which to register the
 authorisation handler", ARG_OPTIONAL, ARG_HAS_VALUE, "c-auth-
handler"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
 * When the authentication service has been registered, this
 function will be
 * called.
 */
```

```
static int
on_registration(SESSION_T *session, void *context)
{
        printf("Registered authentication handler\n");
        return HANDLER_SUCCESS;
}

/**
 * When the authentication service has be deregistered, this
 function will be
 * called.
 */
static int
on_deregistration(SESSION_T *session, void *context)
{
        printf("Deregistered authentication handler\n");
        return HANDLER_SUCCESS;
}

/**
 * This is the function that is called when authentication has
 been delegated
 * from Diffusion.
 *
 * The response may return one of three values via the response
 parameter:
 * ALLOW:   The user is authenticated.
 * ALLOW_WITH_RESULT: The user is authenticated, and additional
 roles are
 *                    to be applied to the user.
 * DENY:    The user is NOT authenticated.
 * ABSTAIN: Allow another handler to make the decision.
 *
 * The handler should return HANDLER_SUCCESS in all cases,
 unless an actual
 * error occurs during the authentication process (in which
 case,
 * HANDLER_FAILURE is appropriate).
 */
static int
on_authentication(SESSION_T *session,
                  const SVC_AUTHENTICATION_REQUEST_T *request,
                  SVC_AUTHENTICATION_RESPONSE_T *response,
                  void *context)
{
        int auth_decided = 0;

        // No credentials, or not password type. We're not an
 authority for
        // this type of authentication so abstain in case some
 other registered
        // authentication handler can deal with the request.
        if(request->credentials == NULL) {
                printf("No credentials specified, abstaining
\n");
                response->value = AUTHENTICATION_ABSTAIN;
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }
        if(request->credentials->type != PLAIN_PASSWORD) {
                printf("Credentials are not PLAIN_PASSWORD,
 abstaining\n");
                response->value = AUTHENTICATION_ABSTAIN;
```

```
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }

        printf("principal = %s\n", request->principal);
        printf("credentials = %*s\n",
               (int)request->credentials->data->len,
               request->credentials->data->data);

        if(request->principal == NULL || strlen(request-
>principal) == 0) {
                printf("Denying anonymous connection (no
 principal)\n");
                response->value = AUTHENTICATION_DENY; // Deny
 anon connections
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }

        char *password = malloc(request->credentials->data->len
 + 1);
        memmove(password, request->credentials->data->data,
 request->credentials->data->len);
        password[request->credentials->data->len] = '\0';

        int i = 0;
        while(USERS[i].username != NULL) {

                printf("Checking username %s vs %s\n", request-
>principal, USERS[i].username);
                printf("     and password %s vs %s\n", password,
 USERS[i].password);

                if(strcmp(USERS[i].username, request->principal)
 == 0 &&
                   strcmp(USERS[i].password, password) == 0) {

                        puts("Allowed");
                        response->value = AUTHENTICATION_ALLOW;
                        auth_decided = 1;
                        break;
                }
                i++;

        }

        free(password);

        if(auth_decided == 0) {
                puts("Abstained");
                response->value = AUTHENTICATION_ABSTAIN;
                auth_decided = 1;
        }

        return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if (options == NULL || hash_get(options, "help") !=
 NULL) {
```

```
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        char *name = hash_get(options, "name");
        char *principal = hash_get(options, "principal");
        char *credentials = hash_get(options, "credentials");

        // Create a session with Diffusion.
        puts("Creating session");
        DIFFUSION_ERROR_T error;
        SESSION_T *session = session_create(url,
                                            principal,
                                            credentials !=
 NULL ? credentials_create_password(credentials) : NULL,
                                            NULL, NULL,
                                            &error);
        if (session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        // Provide a set (via a hash map containing keys and
 NULL values)
        // to indicate what information about the connecting
 client that we'd
        // like Diffusion to send us.
        HASH_T *detail_set = hash_new(5);
        char buf[2];
        sprintf(buf, "%d", SESSION_DETAIL_SUMMARY);
        hash_add(detail_set, strdup(buf), NULL);
        sprintf(buf, "%d", SESSION_DETAIL_LOCATION);
        hash_add(detail_set, strdup(buf), NULL);
        sprintf(buf, "%d", SESSION_DETAIL_CONNECTOR_NAME);
        hash_add(detail_set, strdup(buf), NULL);

        AUTHENTICATION_REGISTRATION_PARAMS_T
 auth_registration_params = {
                .name = name,
                .detail_set = detail_set,
                .on_registration = on_registration,
                .authentication_handlers.on_authentication =
 on_authentication
        };

        // Register the authentication handler.
        puts("Sending registration request");
        SVC_AUTHENTICATION_REGISTER_REQUEST_T *reg_request =
                authentication_register(session,
 auth_registration_params);

        // Wait a while before moving on to deregistration.
        sleep(10);

        AUTHENTICATION_DEREGISTRATION_PARAMS_T
 auth_deregistration_params = {
                .on_deregistration = on_deregistration,
                .original_request = reg_request
        };
```

```
        printf("Deregistering authentication handler\n");
        authentication_deregister(session,
 auth_deregistration_params);

        // Never exit
        while (1) {
                sleep(10);
        }

        // Not called, but this is the way you would gracefully
 terminate the
        // connection with Diffusion.
        session_close(session, &error);

        return(EXIT_SUCCESS);
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Developing a control authentication handler
Implement the `ControlAuthenticationHandler` interface to create a control authentication handler.

About this task

A control authentication handler can be implemented in any language where the Diffusion Unified API includes the AuthenticationControl feature.

For more information, see AuthenticationControl on page 300.

This example demonstrates how to implement a control authentication handler in Java.

> **Note:** Where *c.p.d* is used in package names, it indicates `com.pushtechnology.diffusion`.

Procedure

1. Edit the `etc/Server.xml` configuration file to include a name that the control authentication handler can register with.

   Include the `control-authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `handler-name` attribute is the name that your control authentication handler registers as. For example:

   ```
   <security>
     <authentication-handlers>
       <-- Include a local authentication handler that can
    authenticate the control client -->
       <authentication-handler class="com.example.LocalHandler" /
   >

       <-- Register your control authentication handler -->
       <control-authentication-handler handler-name="before-
   system-handler" />

     </authentication-handlers>
   </security>
   ```

   The client that registers your control authentication handler must first authenticate with the Diffusion server. Configure a local authentication handler that allows the client to connect.
2. Start the Diffusion server.

- On UNIX®-based systems, run the `diffusion.sh` command in the *`diffusion_installation_dir`*`/bin` directory.
- On Windows systems, run the `diffusion.bat` command in the *`diffusion_installation_dir`*`\bin` directory.

3. Create a Java class that implements `ControlAuthenticationHandler`.

```
package com.example.client;

import
 com.pushtechnology.diffusion.client.details.SessionDetails;
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticat
import com.pushtechnology.diffusion.client.types.Credentials;

public class ExampleControlAuthenticationHandler implements
 ControlAuthenticationHandler{

    public void authenticate(String principal, Credentials
 credentials,
              SessionDetails sessionDetails, Callback callback)
 {

        // Logic to make the authentication decision.

        // Authentication decision
        callback.abstain();

        // callback.deny();
        // callback.allow();

    }

    @Override
    public void onActive(RegisteredHandler handler) {

    }

    @Override
    public void onClose() {

    }


}
```

a) Ensure that you import `Credentials` from the *`c.p.d`*`.client.types` package, not the *`c.p.d`*`.api` package.
b) Implement the `authenticate` method.
c) Use the `allow`, `deny`, or `abstain` method on the `Callback` object to respond with the authentication decision.
d) You can override the `onActive` and `onClose` to include actions the control authentication handler performs when the client opens its connection to the Diffusion server and when the client closes its session with the Diffusion server.

   For example, when the client session becomes active, the control authentication handler uses the `onActive` method to open a connection to a database. When the client session is closed, the control authentication handler uses the `onClose` method to close the connection to the database.

4. Create a simple client that registers your control authentication handler with the Diffusion server.

```
package com.example.client;

import com.example.client.ExampleControlAuthenticationHandler;
import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.details.SessionDetails.DetailType;
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticat
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.session.SessionFactory;

import java.util.EnumSet;

public class ExampleControlClient {

    public static void main(String[] args) {

        final Session session;

        // Create the client session
        SessionFactory sf = Diffusion.sessions();
        session = sf.principal("ControlClient1")
                    .passwordCredentials("Passw0rd")
                    .open("ws://diffusion.example.com:80");

        // Get the AuthenticationControl feature
        AuthenticationControl authControl =
 session.feature(AuthenticationControl.class);

        // Use the AuthenticationControl feature to register
 your control authentication
        // handler with the name that you configured in
 Server.xml
        authControl.setAuthenticationHandler("before-system-
handler",
                    EnumSet.allOf(DetailType.class), new
 ExampleControlAuthenticationHandler());

    }
}
```

a) Create a session.

   Change the URL from that provided in the example to the URL of the Diffusion server.

b) Use the session to get the `AuthenticationControl` feature.

c) Use the `AuthenticationControl` feature to register your control authentication handler, `ExampleControlAuthenticationHandler`, using the name that you configured in the `etc/Server.xml` configuration file, `before-system-handler`.

5. Start your client.

   It connects to the Diffusion server and registers the control authentication handler with the name `before-system-handler`.

Results

When a client authenticates, the Diffusion server forwards the authentication request to the authentication handler you have registered. Your authentication handler can ALLOW, DENY, or ABSTAIN from the authentication decision. If your authentication handler returns

an ALLOW or DENY decision, this decision is used as the response to the authenticating client. If your authentication handler returns an ABSTAIN decision, the Diffusion server forwards the authentication request to the next authentication handler. For more information, see Authentication on page 142.

Related Links

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Authentication on page 142
You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Authentication on page 142
You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

Developing a composite control authentication handler

Extend the `CompositeControlAuthenticationHandler` class to combine the decisions from multiple control authentication handlers.

About this task

Using a composite control authentication handler reduces the number of messages that are sent between the Diffusion server and the client to perform authentication.

This example describes how to use a composite control authentication handler as part of a client remote from the Diffusion server.

Procedure

1. Edit the `etc/Server.xml` configuration file to point to your composite control authentication handler.

   Include the `control-authentication-handler` element in the list of authentication handlers. The order of the list defines the order in which the authentication handlers are called. The value of the `handler-name` attribute is the name that your composite control authentication handler registers as. For example:

```
<security>
  <authentication-handlers>
    <-- Include a local authentication handler that can
 authenticate the control client -->
    <authentication-handler class="com.example.LocalHandler" /
>
```

```
    <-- Register your composite control authentication handler
 -->
    <control-authentication-handler handler-name="example-
composite-control-authentication-handler" />

  </authentication-handlers>
</security>
```

The client that registers your control authentication handler must first authenticate with the Diffusion server. Configure a local authentication handler that allows the client to connect.

2. Start the Diffusion server.
   - On UNIX-based systems, run the `diffusion.sh` command in the *`diffusion_installation_dir`*`/bin` directory.
   - On Windows systems, run the `diffusion.bat` command in the *`diffusion_installation_dir`*`\bin` directory.
3. Create the individual control authentication handlers that your composite control authentication handler calls.

   You can follow steps in the task Developing a control authentication handler on page 307.

   In this example, the individual control authentication handlers are referred to as `HandlerOne`, `HandlerTwo`, and `HandlerThree`.
4. Extend the `CompositeControlAuthenticationHandler` class.

```
package com.example.client;

import com.example.client.HandlerOne;
import com.example.client.HandlerTwo;
import com.example.client.HandlerThree;

import
 com.pushtechnology.diffusion.client.features.control.clients.CompositeCo

public class ExampleHandler extends
 CompositeControlAuthenticationHandler {

    public ExampleHandler() {
        super(new HandlerOne(), new HandlerTwo(), new
 HandlerThree());
    }

}
```

   a) Import your individual control authentication handlers.
   b) Create a no-argument constructor that calls the super class constructor with a list of your individual handlers.
5. Create a simple client that registers your composite control authentication handler with the Diffusion server.

   You can follow steps in the task Developing a control authentication handler on page 307.

   Ensure that you register your composite control authentication handler, `ExampleHandler`, using the name that you configured in the `etc/Server.xml` configuration file, `example-composite-control-authentication-handler`.
6. Start your client.

   It connects to the Diffusion server and registers the composite control authentication handler.

Results

When the client session starts, the composite control authentication handler calls the `onActive` methods of the individual control authentication handlers in the order in which they are passed in to the composite handler.

When the composite control authentication handler is called, it calls the individual control authentication handlers that are passed to it as parameters in the order they are passed in.

- If an individual handler responds with ALLOW, the composite handler responds with that decision to the Diffusion server and a list of any roles to assign to the authenticated principal.
- If an individual handler responds with DENY, the composite handler responds with that decision to the Diffusion server.
- If an individual handler responds with ABSTAIN, the composite handler calls the next individual handler in the list.
- If all individual handlers respond with ABSTAIN, the composite handler responds to the Diffusion server with an ABSTAIN decision.

When the client session closes, the composite control authentication handler calls the `onClose` methods of the individual control authentication handlers in the order in which they are passed in to the composite handler.

Related Links

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Authentication on page 142
You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Authentication on page 142
You can implement and register handlers to authenticate clients when the clients try to perform operations that require authentication.

AuthenticationControl on page 300
Use the AuthenticationControl feature to enable a client session to authenticate other clients.

SystemAuthenticationControl

Use the SystemAuthenticationControl feature to enable a client session to update the system authentication store. Principals and their associated credentials and roles are stored in the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users.

The SystemAuthenticationControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
SystemAuthenticationControl	YES	YES	YES	NO

Related Links

System authentication handler on page 147
Diffusion provides an authentication handler that uses principal, credential, and roles information stored in the Diffusion server to make its authentication decision.

Features on page 169
Units of functionality within the Unified API are exposed as features.

Updating the system authentication store
A client can use the SystemAuthenticationControl feature to update the system authentication store. The information in the system authentication store is used by the system authentication handler to authenticate users and assign roles to them.

Querying the store

Required permissions: view_security

The client can get a snapshot of the current information in the system authentication store. This information is returned as an object model.

Updating the store

Required permissions: modify_security

The client can use a command script to update the system authentication store. The command script is a string that contains a command on each line. These commands are applied to the current state of the system authentication store.

The update is transactional. Unless all of the commands in the script can be applied, none of them are.

Using a script builder

You can use a script builder to create the command script used to update the system authentication store. Use the script builder to create commands for the following actions:

- Set the authentication decision for anonymous principals
- Add principals to the store
- Delete principals from the store
- Change the password of a principal
- Assign roles to principals

Related Links

System authentication handler on page 147
Diffusion provides an authentication handler that uses principal, credential, and roles information stored in the Diffusion server to make its authentication decision.

Example: Update the system authentication store
The following examples use the SystemAuthenticationControl feature in the Unified API to update the system authentication store.

JavaScript

Note: Only steps 4 and 5 deal with the system authentication store.

```
// Session security allows you to change the principal that a
 session is authenticated as. It also  allows users to
// query and update server-side security and authentication
 stores, which control users, roles and permissions.
// This enables you to manage the capabilities that any logged
 in user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {

    // 1. A session change their principal by re-authenticating
    session.security.changePrincipal('admin',
 'password').then(function() {
        console.log('Authenticated as admin');
    });

    // 2. The security configuration provides details about
 roles and their assigned permissions

 session.security.getSecurityConfiguration().then(function(config)
 {
        console.log('Roles for anonymous sessions: ',
 config.anonymous);
        console.log('Roles for named sessions: ', config.named);
        console.log('Available roles: ', config.roles);
    }, function(error) {
        console.log('Unable to fetch security configuration',
 error);
    });

    // 3. Changes to the security configuration are done with a
 SecurityScriptBuilder
    var securityScriptBuilder =
 session.security.securityScriptBuilder();

    // Set the permissions for a particular role - global and
 topic-scoped
    // Each method on a script builder returns a new builder
    var setPermissionScript =
 securityScriptBuilder.setGlobalPermissions('SUPERUSER',
 ['REGISTER_HANDLER'])

 .setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
                                                    .build();

    // Update the server-side store with the generated script

 session.security.updateSecurityStore(setPermissionScript).then(function()
 {
        console.log('Security configuration updated
 successfully');
    }, function(error) {
        console.log('Failed to update security configuration: ',
 error);
    });
```

```
    // 4. The system authentication configuration lists all
 users & roles

 session.security.getSystemAuthenticationConfiguration().then(function(conf
 {
        console.log('System principals: ', config.principals);
        console.log('Anonymous sessions: ', config.anonymous);
    }, function(error) {
        console.log('Unable to fetch system authentication
 configuration', error);
    });

    // 5. Changes to the system authentication config are done
 with a SystemAuthenticationScriptBuilder
    var authenticationScriptBuilder =
 session.security.authenticationScriptBuilder();

    // Add a new user and set password & roles.
    var addUserScript =
 authenticationScriptBuilder.addPrincipal('Superman',
 'correcthorsebatterystapler')

 .assignRoles('Superman', ['SUPERUSER'])
                                                    .build();

    // Update the system authentication store
    session.security.updateStore(addUserScript).then(function()
 {
        console.log('Updated system authentication config');
    }, function(error) {
        console.log('Failed to update system authentication: ',
 error);
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;

import java.util.HashSet;
import java.util.Set;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.callbacks.ErrorReason;
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityStore
import com.pushtechnology.diffusion.client.session.Session;
```

```
/**
 * An example of using a control client to alter the system
 authentication
 * configuration.
 * <P>
 * This uses the {@link SystemAuthenticationControl} feature
 only.
 *
 * @author Push Technology Limited
 * @since 5.2
 */
public class ControlClientChangingSystemAuthentication {

    private static final Logger LOG =
        LoggerFactory.getLogger(
            ControlClientChangingSystemAuthentication.class);

    private final SystemAuthenticationControl
 systemAuthenticationControl;

    /**
     * Constructor.
     */
    public ControlClientChangingSystemAuthentication() {

        final Session session = Diffusion.sessions()
            // Authenticate with a user that has the
 VIEW_SECURITY and
            // MODIFY_SECURITY permissions.
            .principal("admin").password("password")
            // Use a secure channel because we're transferring
 sensitive
            // information.
            .open("wss://diffusion.example.com:80");

        systemAuthenticationControl =
            session.feature(SystemAuthenticationControl.class);
    }

    /**
     * For all system users, update the assigned roles to
 replace the
     * "SUPERUSER" role and with "ADMINISTRATOR".
     *
     * @param callback result callback
     */
    public void
 changeSuperUsersToAdministrators(UpdateStoreCallback callback)
 {

        systemAuthenticationControl.getSystemAuthentication(
            new ChangeSuperUsersToAdministrators(callback));
    }

    private final class ChangeSuperUsersToAdministrators
        implements ConfigurationCallback {

        private final UpdateStoreCallback callback;

        ChangeSuperUsersToAdministrators(UpdateStoreCallback
 callback) {
            this.callback = callback;
        }
```

```
        @Override
        public void onReply(SystemAuthenticationConfiguration
 configuration) {

            ScriptBuilder builder =
                systemAuthenticationControl.scriptBuilder();

            // For all system users ...
            for (SystemPrincipal principal :
 configuration.getPrincipals()) {

                final Set<String> assignedRoles =
 principal.getAssignedRoles();

                // ... that have the SUPERUSER assigned role ...
                if (assignedRoles.contains("SUPERUSER")) {
                    final Set<String> newRoles = new
 HashSet<>(assignedRoles);
                    newRoles.remove("SUPERUSER");
                    newRoles.add("ADMINISTRATOR");

                    // ... add a command to the script that
 updates the user's
                    // assigned roles, replacing SUPERUSER with
 "ADMINISTRATOR".
                    builder =
                        builder.assignRoles(principal.getName(),
 newRoles);
                }
            }

            final String script = builder.script();

            LOG.info(
                "Sending the following script to the server:
\n{}",
                script);

            systemAuthenticationControl.updateStore(
                script,
                callback);
        }

        @Override
        public void onError(ErrorReason errorReason) {
            // This might fail if the session lacks the required
 permissions.
            callback.onError(errorReason);
        }
    }

    /**
     * Close the session.
     */
    public void close() {
        systemAuthenticationControl.getSession().close();
    }
}
```

.NET

```
using System.Collections.Generic;
using System.Linq;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients;
using PushTechnology.ClientInterface.Client.Types;
using IUpdateStoreCallback =
 PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityCon

namespace UCIStack.Examples
{
    public class ControlClientChangingSystemAuthentication
    {
        #region Fields

        private readonly ISystemAuthenticationControl
 theSystemAuthenticationControl;

        #endregion Fields

        #region Constructor

        public ControlClientChangingSystemAuthentication()
        {
            var session =
                Diffusion.Sessions
                    // Authenticate with a user that has the
 VIEW_SECURITY and MODIFY_SECURITY permissions

 .Principal( "control" ).Password( "password" )
                    // Use a secure channel because we're
 transferring sensitive information.
                    .Open( "dpt://localhost:8080" );

            theSystemAuthenticationControl =
 session.GetSystemAuthenticationControlFeature();

            session.Start();
        }

        #endregion Constructor

        #region Public Methods

        /// <summary>
        /// For all system users, update the assigned roles to
 replace the 'SUPERUSER' role and with 'ADMINISTRATOR'.
        /// </summary>
        /// <param name="callback"></param>
        public void
 ChangeSuperUsersToAdministrators( IUpdateStoreCallback
 callback )
        {

 theSystemAuthenticationControl.GetSystemAuthentication(
                new
 ChangeSuperusersToAdministrators( theSystemAuthenticationControl,
 callback ) );
        }
```

```
        #endregion Public Methods

        #region Private Classes

        private class InternalUpdateStoreCallback :
IUpdateStoreCallback
        {
            /// <summary>
            /// The script was applied successfully.
            /// </summary>
            public void OnSuccess()
            {
            }

            /// <summary>
            /// The script was rejected.  No changes were made
to the system authentication store.
            /// </summary>
            /// <param name="errors">The details of why the
script was rejected.</param>
            public void OnRejected( ICollection<IErrorReport>
errors )
            {
            }

            /// <summary>
            /// Notification of a contextual error related to
this callback. This is
            /// analogous to an exception being raised.
Situations in which
            /// <code>OnError</code> is called include the
session being closed, a
            /// communication timeout, or a problem with the
provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
            }
        }

        private class ChangeSuperusersToAdministrators :
IConfigurationCallback
        {
            #region Fields

            private readonly ISystemAuthenticationControl
theSystemAuthenticationControl;
            private readonly IUpdateStoreCallback theCallback;

            #endregion Fields

            #region Constructor

            public ChangeSuperusersToAdministrators(
                ISystemAuthenticationControl
systemAuthenticationControl,
                IUpdateStoreCallback callback )
```

```
            {
                theSystemAuthenticationControl =
systemAuthenticationControl;
                theCallback = callback;
            }

            #endregion Constructor

            /// <summary>
            /// The configuration callback reply.
            /// </summary>
            /// <param
name="systemAuthenticationConfiguration">The system
authenticationConfiguration stored on the server.</param>
            public void
OnReply( ISystemAuthenticationConfiguration
systemAuthenticationConfiguration )
            {
                var builder =
theSystemAuthenticationControl.ScriptBuilder();

                // For all system users...
                foreach( var principal in
systemAuthenticationConfiguration.Principals )
                {
                    var assignedRoles = principal.AssignedRoles;

                    // ...that have the 'SUPERUSER' assigned
role...
                    if( !assignedRoles.Contains( "SUPERUSER" ) )
continue;

                    var newRoles = new
HashSet<string>( assignedRoles );

                    newRoles.Remove( "SUPERUSER" );
                    newRoles.Add( "ADMINISTRATOR" );

                    // ...and add a command to the script that
updates the user's assigned roles, replacing 'SUPERUSER' with
                    // 'ADMINISTRATOR'.
                    builder =
builder.AssignRoles( principal.Name, newRoles.ToList() );
                }

                var script = builder.Script();


theSystemAuthenticationControl.UpdateStore( script,
theCallback );
            }

            /// <summary>
            /// Notification of a contextual error related to
this callback. This is
            /// analogous to an exception being raised.
Situations in which
            /// <code>OnError</code> is called include the
session being closed, a
            /// communication timeout, or a problem with the
provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
```

```
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
                theCallback.OnError( errorReason );
            }
        }

        #endregion Private Classes

        #region Tests

        //[Test]
        public void
 ControlClientChangingSystemAuthenticationTest()
        {
            var client = new
 ControlClientChangingSystemAuthentication();

            client.ChangeSuperUsersToAdministrators( new
 InternalUpdateStoreCallback() );
        }

        #endregion Tests
    }
}
```

C

```
#include <stdio.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "service/svc-system-auth-control.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

int
on_get_system_authentication_store(SESSION_T *session,
                                   const
 SYSTEM_AUTHENTICATION_STORE_T store,
                                   void *context)
```

```
{
        puts("on_get_system_authentication_store()");

        printf("Got %ld principals\n", store.system_principals-
>size);

        char **names = get_principal_names(store);
        for(char **name = names; *name != NULL; name++) {
                printf("Principal: %s\n", *name);

                char **roles = get_roles_for_principal(store,
 *name);
                for(char **role = roles; *role != NULL; role++)
 {
                    printf("  |- Role: %s\n", *role);
                }
                free(roles);
        }
        free(names);

        switch(store.anonymous_connection_action) {
        case ANONYMOUS_CONNECTION_ACTION_ALLOW:
                puts("Allow anonymous connections");
                break;
        case ANONYMOUS_CONNECTION_ACTION_DENY:
                puts("Deny anonymous connections");
                break;
        case ANONYMOUS_CONNECTION_ACTION_ABSTAIN:
                puts("Abstain from making anonymous connection
 decision");
                break;
        }

        puts("Anonymous connection roles:");
        char **roles = get_anonymous_roles(store);
        for(char **role = roles; *role != NULL; role++) {
                printf("  |- Role: %s\n", *role);
        }
        free(roles);

        apr_thread_cond_broadcast(cond);

        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
        // Standard command line parsing.
        const HASH_T *options = parse_cmdline(argc, argv,
 arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        const char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
```

```
                credentials =
 credentials_create_password(password);
        }

        // Setup for condition variable
        apr_initialize();
        apr_pool_create(&pool, NULL);
        apr_thread_mutex_create(&mutex,
 APR_THREAD_MUTEX_DEFAULT, pool);
        apr_thread_cond_create(&cond, pool);

        // Setup for session
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, principal, credentials,
 NULL, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        const GET_SYSTEM_AUTHENTICATION_STORE_PARAMS_T params =
 {
                .on_get = on_get_system_authentication_store
        };

        apr_thread_mutex_lock(mutex);

        get_system_authentication_store(session, params);

        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        return 0;
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

DSL syntax: system authentication store

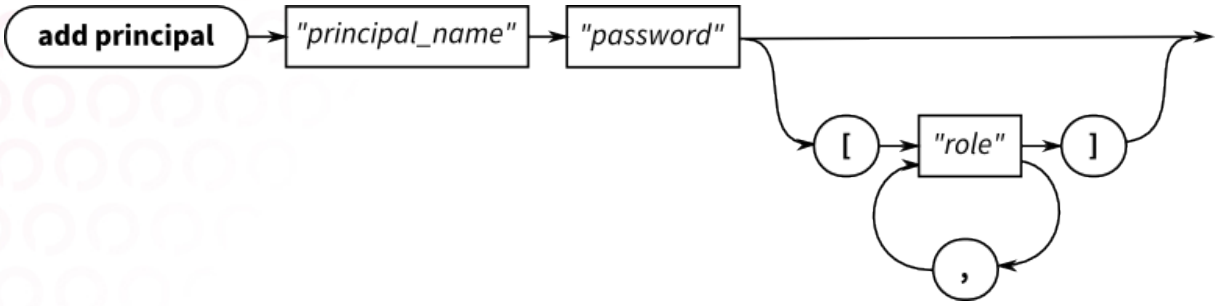
The scripts that you can use with the SystemAuthenticationControl feature to update the system authentication store are formatted according to a domain-specific language (DSL).

You can use the script builders provided in the APIs to create a script to update the system authentication store. However, if you want to create the script by some other method, ensure that it conforms to the DSL.

The following sections each describe the syntax for a single line of the script file.

Adding a principal

Railroad diagram



Backus-Naur form

`add principal` *"principal_name"* *"password"* `[ [` *"role"* `[` *"role"* `] ]`

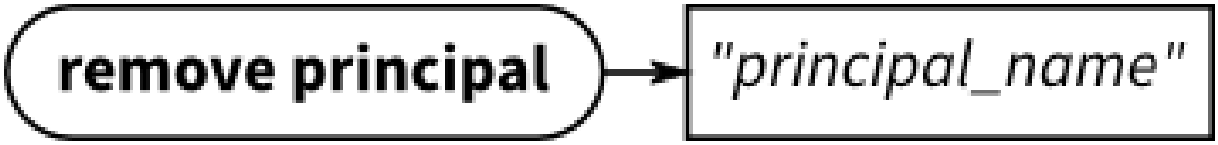
Example

```
add principal "user6" "passw0rd"
add principal "user13" "passw0rd" ["CLIENT" "TOPIC_CONTROL"]
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Removing a principal

Railroad diagram



Backus-Naur form

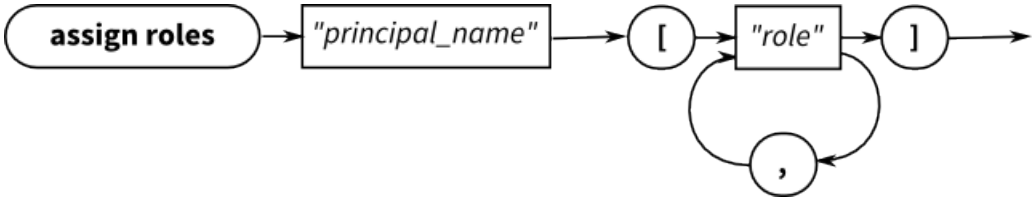
`remove principal` *"principal_name"*

Example

```
remove principal "user25"
```

Assigning roles to a principal

Railroad diagram



Backus-Naur form

`assign roles` *"principal_name"* `[` *"role"* `[ ,` *"role"* `] ]`

Example

```
assign roles "agent77" ["CLIENT", "CLIENT_CONTROL"]
```

When you use this command to assign roles to a principal, it overwrites any existing roles assigned to that principal. Ensure that all the roles you want the principal to have are listed in the command.

Setting the password for a principal

Railroad diagram

Backus-Naur form

`set password` "*principal_name*" "*password*"

Example

```
set password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Verifying the password for a principal

Railroad diagram

Backus-Naur form

`verify password` "*principal_name*" "*password*"

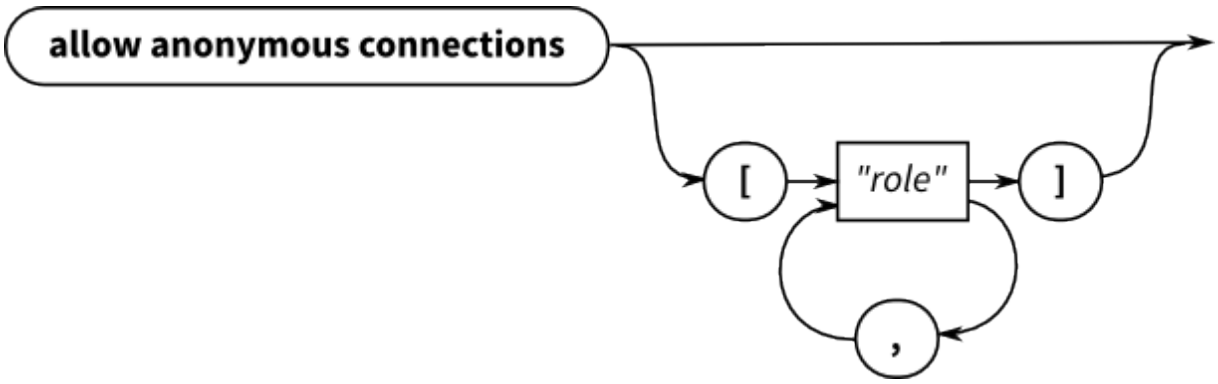
Example

```
verify password "user1" "passw0rd"
```

The password is passed in as plain text, but is stored in the system authentication store as a secure hash.

Allowing anonymous connections

Railroad diagram



Backus-Naur form

`allow anonymous connections [ [ "`*role*`" [ , "`*role*`" ] ] ]`

Example

```
allow anonymous connections [ "CLIENT" ]
```

Denying anonymous connections

Railroad diagram

Backus-Naur form

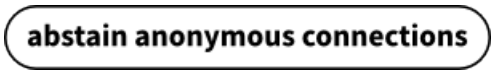
```
deny anonymous connections
```

Example

```
deny anonymous connections
```

Abstaining from providing a decision about anonymous connections

Railroad diagram



Backus-Naur form

```
abstain anonymous connections
```

Example

```
abstain anonymous connections
```

SecurityControl

Use the SecurityControl feature to enable a client to update the security store. Roles and their associated permissions are defined in the security store. The information in the security store is used by the Diffusion server to decide whether a client session with a set of roles has the required permissions to perform an action.

The SecurityControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
SecurityControl	YES	YES	YES	NO

Updating the security store

A client can use the SecurityControl feature to update the security store. The information in the security store is used by the Diffusion server to define the permissions assigned to roles and the roles assigned to anonymous sessions and named sessions.

Querying the store

Required permissions: view_security

The client can get a snapshot of the current information in the security store. This information is returned as an object model.

Updating the store

Required permissions: modify_security

The client can use a command script to update the security store. The command script is a string that contains a command on each line. These commands are applied to the current state of the security store.

The update is transactional. Unless all of the commands in the script can be applied, none of them are.

Using a script builder

You can use a script builder to create the command script used to update the security store. Use the script builder to create commands for the following actions:

- Set the global permissions assigned to a named role
- Set the default topic permissions assigned to a named role
- Set the topic permissions associated with a specific topic path assigned to a named role

  This can include explicitly setting a role to have no permissions at a topic path.
- Remove the topic permissions associated with a specific topic path assigned to a named role
- Set the roles included in a named role
- Set the roles assigned to sessions authenticated with a named principal
- Set the roles assigned to anonymous sessions

Example: Update the security store
The following examples use the SecurityControl feature in the Unified API to update the security store.

JavaScript

Note: Only steps 2 and 3 deal with the security store.

```
// Session security allows you to change the principal that a
 session is authenticated as. It also  allows users to
// query and update server-side security and authentication
 stores, which control users, roles and permissions.
// This enables you to manage the capabilities that any logged
 in user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {

    // 1. A session change their principal by re-authenticating
    session.security.changePrincipal('admin',
 'password').then(function() {
        console.log('Authenticated as admin');
    });

    // 2. The security configuration provides details about
 roles and their assigned permissions

 session.security.getSecurityConfiguration().then(function(config)
 {
        console.log('Roles for anonymous sessions: ',
 config.anonymous);
        console.log('Roles for named sessions: ', config.named);
        console.log('Available roles: ', config.roles);
    }, function(error) {
        console.log('Unable to fetch security configuration',
 error);
    });

    // 3. Changes to the security configuration are done with a
 SecurityScriptBuilder
```

```
    var securityScriptBuilder =
session.security.securityScriptBuilder();

    // Set the permissions for a particular role - global and
topic-scoped
    // Each method on a script builder returns a new builder
    var setPermissionScript =
securityScriptBuilder.setGlobalPermissions('SUPERUSER',
['REGISTER_HANDLER'])

.setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
                                                    .build();

    // Update the server-side store with the generated script

session.security.updateSecurityStore(setPermissionScript).then(function()
{
        console.log('Security configuration updated
successfully');
    }, function(error) {
        console.log('Failed to update security configuration: ',
error);
    });

    // 4. The system authentication configuration lists all
users & roles

session.security.getSystemAuthenticationConfiguration().then(function(conf
{
        console.log('System principals: ', config.principals);
        console.log('Anonymous sessions: ', config.anonymous);
    }, function(error) {
        console.log('Unable to fetch system authentication
configuration', error);
    });

    // 5. Changes to the system authentication config are done
with a SystemAuthenticationScriptBuilder
    var authenticationScriptBuilder =
session.security.authenticationScriptBuilder();

    // Add a new user and set password & roles.
    var addUserScript =
authenticationScriptBuilder.addPrincipal('Superman',
'correcthorsebatterystapler')

.assignRoles('Superman', ['SUPERUSER'])
                                                    .build();

    // Update the system authentication store
    session.security.updateStore(addUserScript).then(function()
{
        console.log('Updated system authentication config');
    }, function(error) {
        console.log('Failed to update system authentication: ',
error);
    });
});
```

Java

```
package com.pushtechnology.diffusion.examples;
```

```
import java.util.Collections;
import java.util.Map;
import java.util.Set;
import java.util.TreeSet;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.callbacks.ErrorReason;
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityContr
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityContr
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityContr
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityContr
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityContr
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityStore
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.types.GlobalPermission;
import
 com.pushtechnology.diffusion.client.types.TopicPermission;

/**
 * An example of using a control client to alter the security
 configuration.
 * <P>
 * This uses the {@link SecurityControl} feature only.
 *
 * @author Push Technology Limited
 * @since 5.3
 */
public class ControlClientChangingSecurity {

    private static final Logger LOG =
        LoggerFactory.getLogger(
            ControlClientChangingSecurity.class);

    private final SecurityControl securityControl;

    /**
     * Constructor.
     */
    public ControlClientChangingSecurity() {

        final Session session = Diffusion.sessions()
            // Authenticate with a user that has the
 VIEW_SECURITY and
            // MODIFY_SECURITY permissions.
            .principal("admin").password("password")
            // Use a secure channel because we're transferring
 sensitive
            // information.
            .open("wss://diffusion.example.com:80");
```

```
        securityControl =
session.feature(SecurityControl.class);
    }

    /**
     * This will update the security store to ensure that all
roles start with a
     * capital letter (note that this does not address changing
the use of the
     * roles in the system authentication store).
     *
     * @param callback result callback
     */
    public void capitalizeRoles(UpdateStoreCallback callback) {
        securityControl.getSecurity(new
CapitalizeRoles(callback));
    }

    private final class CapitalizeRoles implements
ConfigurationCallback {

        private final UpdateStoreCallback callback;

        CapitalizeRoles(UpdateStoreCallback callback) {
            this.callback = callback;
        }

        @Override
        public void onReply(SecurityConfiguration configuration)
{

            ScriptBuilder builder =
                securityControl.scriptBuilder();

            builder = builder.setRolesForAnonymousSessions(

capitalize(configuration.getRolesForAnonymousSessions()));

            builder = builder.setRolesForNamedSessions(

capitalize(configuration.getRolesForNamedSessions()));

            for (Role role : configuration.getRoles()) {

                final String oldName = role.getName();
                final String newName = capitalize(oldName);

                // Only if new name is different
                if (!oldName.equals(newName)) {

                    // Global Permissions
                    final Set<GlobalPermission>
globalPermissions =
                        role.getGlobalPermissions();
                    if (!globalPermissions.isEmpty()) {
                        // Remove global permissions for old
role
                        builder =
                            builder.setGlobalPermissions(
                                oldName,

Collections.<GlobalPermission>emptySet());
                        // Set global permissions for new role
```

```
                        builder =
                            builder.setGlobalPermissions(
                                newName,
                                role.getGlobalPermissions());
                    }

                    final Set<TopicPermission>
defaultTopicPermissions =
                        role.getDefaultTopicPermissions();
                    if (!defaultTopicPermissions.isEmpty()) {
                        // Remove default topic permissions for
old role
                        builder =
                            builder.setDefaultTopicPermissions(
                                oldName,

Collections.<TopicPermission>emptySet());
                        // Set default topic permissions for new
role
                        builder =
                            builder.setDefaultTopicPermissions(
                                newName,

role.getDefaultTopicPermissions());
                    }

                    final Map<String, Set<TopicPermission>>
topicPermissions =
                        role.getTopicPermissions();

                    if (!topicPermissions.isEmpty()) {
                        for (Map.Entry<String,
Set<TopicPermission>> entry : topicPermissions
                            .entrySet()) {
                            final String topicPath =
entry.getKey();
                            // Remove old topic permissions
                            builder =
                                builder.removeTopicPermissions(
                                    oldName,
                                    topicPath);
                            // Set new topic permissions
                            builder =
                                builder.setTopicPermissions(
                                    newName,
                                    topicPath,
                                    entry.getValue());
                        }
                    }

                }

                final Set<String> oldIncludedRoles =
role.getIncludedRoles();
                if (!oldIncludedRoles.isEmpty()) {

                    if (!oldName.equals(newName)) {
                        // Remove old included roles
                        builder =
                            builder.setRoleIncludes(
                                oldName,
                                Collections.<String>emptySet());
                    }
```

```
                    // This is done even if role name did not
 change as it is
                    // possible that roles included may have
                    final Set<String> newIncludedRoles =
                        capitalize(oldIncludedRoles);
                    builder =
                        builder.setRoleIncludes(
                            newName,
                            newIncludedRoles);

                }


            }

            final String script = builder.script();

            LOG.info(
                "Sending the following script to the server:
\n{}",
                script);

            securityControl.updateStore(
                script,
                callback);
        }

        private Set<String> capitalize(Set<String> roles) {
            final Set<String> newSet = new TreeSet<>();
            for (String role : roles) {
                newSet.add(capitalize(role));
            }
            return newSet;
        }

        private String capitalize(String role) {
            return Character.toUpperCase(role.charAt(0)) +
 role.substring(1);
        }

        @Override
        public void onError(ErrorReason errorReason) {
            // This might fail if the session lacks the required
 permissions.
            callback.onError(errorReason);
        }
    }

    /**
     * Close the session.
     */
    public void close() {
        securityControl.getSession().close();
    }
}
```

.NET

```
using System.Collections.Generic;
using System.Linq;
using PushTechnology.ClientInterface.Client.Callbacks;
```

```
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityCon
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
    /// <summary>
    /// An example of using a control client to alter the
 security configuration.
    ///
    /// This uses the <see cref="ISecurityControl"/> feature
 only.
    /// </summary>
    public class ControlClientChangingSecurity
    {
        #region Fields

        private readonly ISecurityControl securityControl;

        #endregion Fields

        #region Constructor

        public ControlClientChangingSecurity()
        {
            var session = Diffusion.Sessions
                // Authenticate with a user that has the
 VIEW_SECURITY and MODIFY_SECURITY permissions.
                .Principal( "admin" ).Password( "password" )
                // Use a secure channel because we're
 transferring sensitive information.
                .Open( "wss://diffusion.example.com:80" );

            securityControl =
 session.GetSecurityControlFeature();
        }

        #endregion Constructor

        #region Public Methods

        public void DoCapitalizeRoles( IUpdateStoreCallback
 callback )
        {
            securityControl.GetSecurity( new
 CapitalizeRoles( securityControl, callback ) );
        }

        #endregion Public Methods

        #region Private Classes

        private class CapitalizeRoles : IConfigurationCallback
        {
            #region Fields

            private readonly ISecurityControl
 theSecurityControl;
            private readonly IUpdateStoreCallback theCallback;

            #endregion Fields
```

```
            #region Constructor

            /// <summary>
            /// Constructor.
            /// </summary>
            /// <param name="securityControl">The security
 control object.</param>
            /// <param name="callback">The callback object.</
param>
            public CapitalizeRoles( ISecurityControl
 securityControl, IUpdateStoreCallback callback )
            {
                theSecurityControl = securityControl;
                theCallback = callback;
            }

            #endregion Constructor

            /// <summary>
            /// Notification of a contextual error related to
 this callback. This is
            /// analogous to an exception being raised.
 Situations in which
            /// <code>OnError</code> is called include the
 session being closed, a
            /// communication timeout, or a problem with the
 provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
                // This might fail if the session lacks the
 required permissions.
                theCallback.OnError( errorReason );
            }

            /// <summary>
            /// This is called to return the requested security
 configuration.
            /// </summary>
            /// <param name="configuration">The snapshot of
 information from the security store.</param>
            public void OnReply( ISecurityConfiguration
 configuration )
            {
                var builder =
 theSecurityControl.ScriptBuilder();

                builder = builder.SetRolesForAnonymousSessions(

 Capitalize( configuration.RolesForAnonymousSessions ) );

                builder = builder.SetRolesForNamedSessions(

 Capitalize( configuration.RolesForNamedSessions ) );

                foreach( var role in configuration.Roles )
                {
```

```
                    var oldName = role.Name;
                    var newName = Capitalize( oldName );

                    // Only if new name is different
                    if( !oldName.Equals( newName ) )
                    {
                        // Global permissions
                        var globalPermissions =
role.GlobalPermissions;

                        if( globalPermissions.Count > 0 )
                        {
                            // Remove global permissions for old
role
                            builder =
builder.SetGlobalPermissions( oldName, new
List<GlobalPermission>() );

                            // Set global permissions for new
role
                            builder =
builder.SetGlobalPermissions( newName,
                                new
List<GlobalPermission>( role.GlobalPermissions ) );
                        }

                        var defaultTopicPermissions =
role.DefaultTopicPermissions;

                        if( defaultTopicPermissions.Count > 0 )
                        {
                            // Remove default topic permissions
for old role
                            builder =
builder.SetDefaultTopicPermissions( oldName, new
List<TopicPermission>() );

                            // Set default topic permissions for
new role
                            builder =
builder.SetDefaultTopicPermissions( newName,
                                new
List<TopicPermission>( role.DefaultTopicPermissions ) );
                        }

                        var topicPermissions =
role.TopicPermissions;

                        if( topicPermissions.Count > 0 )
                        {
                            foreach( var entry in
topicPermissions )
                            {
                                var topicPath = entry.Key;

                                // Remove old topic permissions
                                builder =
builder.RemoveTopicPermissions( oldName, topicPath );

                                // Set new topic permissions
                                builder =
builder.SetTopicPermissions( newName, topicPath, entry.Value );
                            }
```

```
                    }
                }

                var oldIncludedRoles = role.IncludedRoles;

                if( oldIncludedRoles.Count > 0 )
                {
                    // Remove old included roles
                    builder =
builder.SetRoleIncludes( oldName, new List<string>() );
                }

                // This is done even if role name did not
change as it is possible that roles included may have
                var newIncludedRoles =
Capitalize( oldIncludedRoles );

                builder = builder.SetRoleIncludes( newName,
newIncludedRoles );
            }
        }

        #region Private Methods

        private static List<string>
Capitalize( IEnumerable<string> roles )
        {
            return roles.Select( Capitalize ).ToList();
        }

        private static string Capitalize( string role )
        {
            return char.ToUpper( role[0] ) +
role.Substring( 1 );
        }

        #endregion Private Methods
    }

    #endregion Private Classes
  }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

DSL syntax: security store

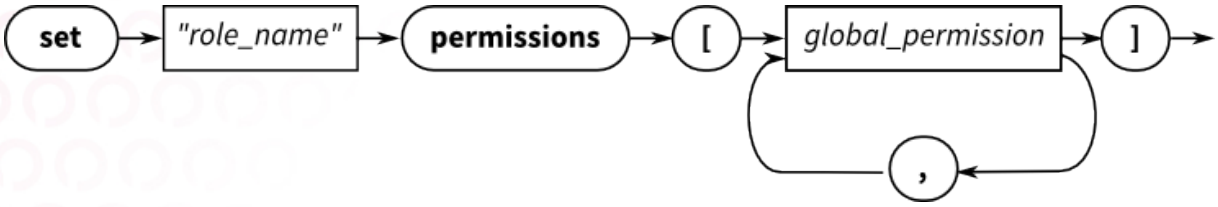
The scripts that you can use with the SecurityControl feature to update the security store are formatted according to a domain-specific language (DSL).

You can use the script builders provided in the APIs to create a script to update the security store. However, if you want to create the script by some other method, ensure that it conforms to the DSL.

The following sections each describe the syntax for a single line of the script file.

Assigning global permissions to a role

Railroad diagram



Backus-Naur form

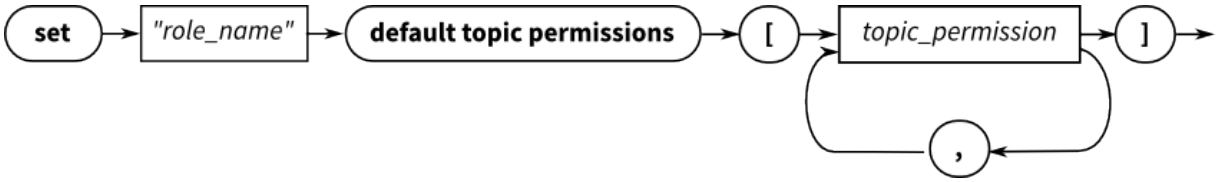
`set "` *role_name* `" permissions [ [` *global_permission* `[ , ` *global_permission* `] ] ]`

Example

```
set "ADMINISTRATOR" permissions [CONTROL_SERVER, VIEW_SERVER,
 VIEW_SECURITY, MODIFY_SECURITY]
set "CLIENT_CONTROL" permissions [VIEW_SESSION, MODIFY_SESSION,
 REGISTER_HANDLER]
```

Assigning default topic permissions to a role

Railroad diagram



Backus-Naur form

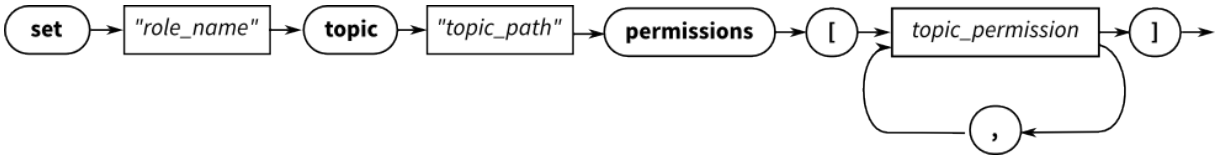
`set "` *role_name* `" default topic permissions [ [` *topic_permission* `[ , ` *topic_permission* `] ] ]`

Example

```
set "CLIENT" default topic permissions [READ_TOPIC ,
 SEND_TO_MESSAGE_HANDLER]
```

Assigning topic permissions associated with a specific topic path to a role

Railroad diagram



Backus-Naur form

`set "` *role_name* `" topic "` *topic_path* `" permissions [ [` *topic_permission* `[ , ` *topic_permission* `] ] ]`

Example

```
set "CLIENT" topic "foo/bar" permissions [READ_TOPIC,
 SEND_TO_MESSAGE_HANDLER]
set "ADMINISTRATOR" topic "foo" permissions [ MODIFY_TOPIC ]
set "CLIENT_CONTROL" topic "foo" permissions [ ]
```

Removing all topic permissions associated with a specific topic path to a role

Railroad diagram

Backus-Naur form

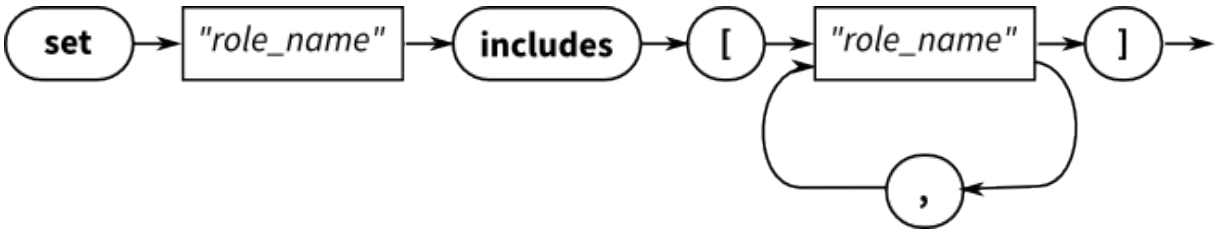
`remove "` *role_name* `" permissions for topic "` *topic_path* `"`

Example

```
remove "CLIENT" permissions for topic "foo/bar"
```

Including roles within another role

Railroad diagram



Backus-Naur form

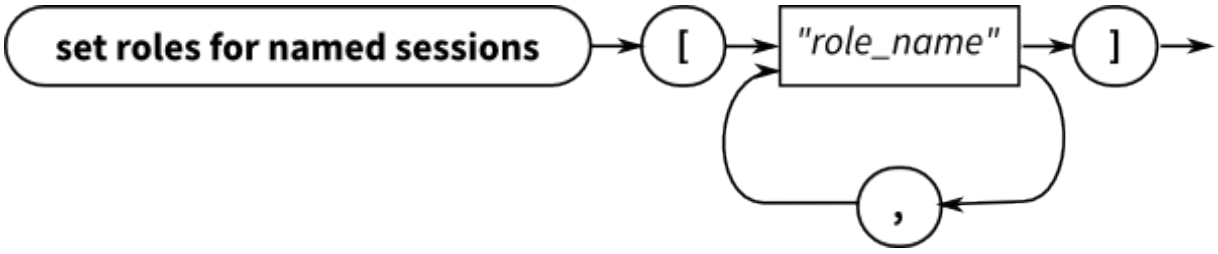
`set "` *role_name* `" includes [ "` *role_name* `" [ , "` *role_name* `" ] ]`

Example

```
set "ADMINISTRATOR" includes ["CLIENT_CONTROL" ,
 "TOPIC_CONTROL"]
set "CLIENT_CONTROL" includes ["CLIENT"]
```

Assigning roles to a named session

Railroad diagram



Backus-Naur form

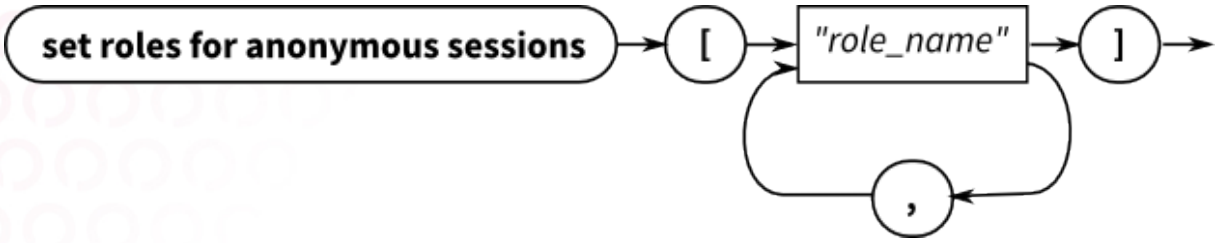
`set roles for named sessions [ "` *role_name* `" [ , "` *role_name* `" ] ]`

Example

```
set roles for named sessions ["CLIENT"]
```

Assigning roles to an anonymous session

Railroad diagram



Backus-Naur form

```
set roles for anonymous sessions[[" role_name "[ ," role_name "]]]
```

Example

```
set roles for anonymous sessions ["CLIENT"]
```

ClientControl

Use the ClientControl feature to enable a client session to receive notifications about other clients and to manage other clients.

The ClientControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
ClientControl	NO	YES	YES	NO

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Managing clients

A client can use the ClientControl feature to manage other client sessions.

Receiving notifications of other client sessions

Required permissions: view_session

A client can set a `SessionDetailsListener` that is notified of all client sessions that open and close.

When the client first registers a listener, it receives a notification for every client session that is currently open.

When the client declares a listener, it can specify exactly the level of detail that the it wants to receive for each client session. The client can request no details, in which case the client receives only session IDs.

When a client is notified of a session closing, it also receives the reason that the session was closed.

Getting details of specific clients

Required permissions: view_session

A client can request details of any client session from the Diffusion server, providing the requesting client knows the session ID of the target client.

Receiving notifications of client queue events

Required permissions: view_session, register_handler

A client can register a `QueueEventHandler` that is notified when outbound client queues at the server reach pre-configured thresholds.

Handling client queue events

Required permissions: view_session, modify_session

A client can respond to a client queue getting full by setting conflation on for the client. A client is also able to set throttling on for specific clients, which also sets conflation.

Closing client sessions

Required permissions: view_session, modify_session

A client can close any client session, providing the requesting client knows the session ID of the target client.

Example: Apply conflation or throttling to a client
The following examples use the Unified API to set queue thresholds and register listeners for queue events.

Java

```
package com.pushtechnology.diffusion.examples;

import static
 com.pushtechnology.diffusion.client.features.control.clients.MessageQueueP

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.MessageQueueP
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;

/**
 * This demonstrates the use of a control client to apply both
 throttling and
 * conflation to clients. It throttles and conflates all clients
 that reach
 * their queue thresholds and remove when they go down again.
 * <P>
 * This uses the 'ClientControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientConflateAndThrottle {

    private final Session session;
    private final ClientControl clientControl;
    private final ClientCallback clientCallback;

    /**
```

```
     * Constructor.
     *
     * @param callback notifies callback from throttle requests
     */
    public ControlClientConflateAndThrottle(ClientCallback
callback) {

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        // Create the ClientControl feature with a handler that
sets queue
        // thresholds on new connecting clients and sets a
listener for queue
        // events.
        clientControl = session.feature(ClientControl.class);
        clientCallback = callback;

        // To register a queue event handler, the client session
must have
        // the 'register handler' and 'view_session'
permissions.
        clientControl.setQueueEventHandler(new
MyThresholdHandler());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

    private class MyThresholdHandler extends
QueueEventHandler.Default {

        @Override
        public void onUpperThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The setThrottled method enables throttling and
conflation.
            // This method requires the client session to have
the
            // 'modify_session' permission.
            clientControl.setThrottled(client, MESSAGE_INTERVAL,
10,
                clientCallback);
        }

        @Override
        public void onLowerThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The setThrottled method enables throttling and
conflation.
            // This method requires the client session to have
the
            // 'modify_session' permission.
```

```
            clientControl
                .setThrottled(client, MESSAGE_INTERVAL, 1000,
 clientCallback);
        }
    }
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Enums;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients;
using PushTechnology.ClientInterface.Client.Session;
using
 PushTechnology.ClientInterface.CommandServices.Commands.Control.Client;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates the use of a control client to apply both
 throttling and conflation to clients.  It throttles
 /// and conflates all clients that reach their queue thresholds
 and remove when they go down again.
 ///
 /// This uses the <see cref="IClientControl"/> feature.
 /// </summary>
 public class ControlClientConflateAndThrottle
 {
  #region Fields

  private readonly ISession session;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  /// <param name="callback">Notifies callback from throttle
 requests.</param>
  public ControlClientConflateAndThrottle( IClientCallback
 callback )
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   // Create the client control feature with a handler that sets
 queue thresholds on new connecting clients
   // and sets a listener for queue events.
   var clientControl = session.GetClientControlFeature();

   // To register a queue event handler, the client session must
 have the REGISTER_HANDLER and VIEW_SESSION
   // permissions.
   clientControl.SetQueueEventHandler( new
 MyThresholdHandler( clientControl, callback ) );
  }
```

```
  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods

  #region Private Classes

  private class MyThresholdHandler : QueueEventHandlerDefault
  {
   #region Fields

   private readonly IClientControl theClientControl;
   private readonly IClientCallback theClientCallback;

   #endregion Fields

   #region Constructor

   public MyThresholdHandler( IClientControl clientControl,
 IClientCallback clientCallback )
   {
    theClientControl = clientControl;
    theClientCallback = clientCallback;
   }

   #endregion Constructor

   #region Overrides

   /// <summary>
   /// The configured upper queue threshold for a client's queue
 has been reached.
   /// </summary>
   /// <param name="client">The client session identifier.</
param>
   /// <param name="policy">The message queue policy.</param>
   public override void OnUpperThresholdCrossed( SessionId
 client, IMessageQueuePolicy policy )
   {
    // The SetThrottled method enables throttling and
 conflation.  This method requires the client session
    // to have the MODIFY_SESSION topic permission.
    theClientControl.SetThrottled( client,
 ThrottlerType.MESSAGE_INTERVAL, 10, theClientCallback );
   }

   /// <summary>
   /// The configured lower threshold for a client's queue has
 been reached.
   /// </summary>
   /// <param name="client">The client session identifier.</
param>
   /// <param name="policy">The message queue policy.</param>
```

```
  public override void OnLowerThresholdCrossed( SessionId
client, IMessageQueuePolicy policy )
  {
   // The SetThrottled method enables throttling and
conflation.  This method requires the client session
   // to have the MODIFY_SESSION topic permission.
   theClientControl.SetThrottled( client,
ThrottlerType.MESSAGE_INTERVAL, 1000, theClientCallback );
  }

  #endregion Overrides
 }

 #endregion Private Classes
}
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

SubscriptionControl

Use the SubscriptionControl feature to enable a client to subscribe other clients to topics and handle routing topic subscription requests.

The SubscriptionControl feature is available in the following APIs:

Feature	JavaScript	Java	.NET	C
SubscriptionControl	NO	YES	YES	NO

Related Links

Features on page 169
Units of functionality within the Unified API are exposed as features.

Managing subscriptions

A client can use the SubscriptionControl feature to subscribe other client sessions to topics that they have not requested subscription to themselves and also to unsubscribe clients from topics. It also enables the client to register as the handler for routing topic subscriptions.

Subscribing and unsubscribing clients

Required permissions: modify_session

A client can subscribe clients that it knows about to topics that those clients have not explicitly requested. It can also unsubscribe clients from topics.

A session identifier is required to specify the client session that is to be subscribed or unsubscribed. Use the ClientControl feature to get the identifiers for connected client sessions.

The SubscriptionControl feature uses topic selectors to specify topics for subscription and unsubscription. Many topics can be specified in a single operation.

The client being subscribed to topics must have read_topic permission for the topics it is being subscribed to.

Acting as a routing subscription handler

Required permissions: view_session, modify_session, register_handler

Routing topics can be created with a server-side handler that assigns clients to real topics. However, you can omit the server-side handler such that subscriptions to routing topics are directed at a client acting as a routing subscription handler.

A client can register a routing subscription handler for a branch of the topic tree. Any subscription requests to routing topics in that branch that do not have server-side handlers are passed to the client for action.

On receipt of a routing subscription request the client can respond with a route request that specifies the path of the actual topic that the routing topic maps to for the requesting client. This subscription fails if the target topic does not already exist or if the requesting client does not have read_topic permission for the routing topic or target topic.

The client can complete other actions before calling back to route. For example, it could use the TopicControl feature to create the topic that the client is to map to.

Alternatively, the client can defer the routing subscription request in which case the requesting client remains unsubscribed. This is similar to denying it from an authorization point of view.

The client must reply with a route or defer for all routing requests.

Related Links

Topic selectors in the Unified API on page 66
A topic selector identifies one or more topics. You can create a topic selector object from a pattern expression.

Topic selectors in the Classic API on page 72
A topic selector is a string that can be used by the Classic API to select more than one topic by indicating that subordinate topics are to be included or by fuzzy matching on topic names or both.

Example: Subscribe other clients to topics
The following examples use the SubscriptionControl feature in the Unified API to subscribe other client sessions to topics.

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.SubscriptionCo
import
 com.pushtechnology.diffusion.client.features.control.topics.SubscriptionCo
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;

/**
 * This demonstrates using a client to subscribe and unsubscribe
 other clients
 * to topics.
 * <P>
 * This uses the 'SubscriptionControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientSubscriptionControl {

    private final Session session;
```

```
    private final SubscriptionControl subscriptionControl;

    /**
     * Constructor.
     */
    public ControlClientSubscriptionControl() {

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        subscriptionControl =
session.feature(SubscriptionControl.class);
    }

    /**
     * Subscribe a client to topics.
     *
     * @param sessionId client to subscribe
     * @param topicSelector topic selector expression
     * @param callback for subscription result
     */
    public void subscribe(
        SessionId sessionId,
        String topicSelector,
        SubscriptionCallback callback) {

        // To subscribe a client to a topic, this client session
        // must have the 'modify_session' permission.
        subscriptionControl.subscribe(
            sessionId,
            topicSelector,
            callback);
    }

    /**
     * Unsubscribe a client from topics.
     *
     * @param sessionId client to unsubscribe
     * @param topicSelector topic selector expression
     * @param callback for unsubscription result
     */
    public void unsubscribe(
        SessionId sessionId,
        String topicSelector,
        SubscriptionCallback callback) {

        // To unsubscribe a client from a topic, this client
session
        // must have the 'modify_session' permission.
        subscriptionControl.unsubscribe(
            sessionId,
            topicSelector,
            callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
```

```
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates using a client to subscribe and
 unsubscribe other clients to topics.
 ///
 /// This uses the <see cref="ISubscriptionControl"/> feature.
 /// </summary>
 public class ControlClientSubscriptionControl
 {
  #region Fields

  private readonly ISession session;
  private readonly ISubscriptionControl subscriptionControl;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  public ControlClientSubscriptionControl()
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   subscriptionControl =
 session.GetSubscriptionControlFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Subscribe a client to topics.
  /// </summary>
  /// <param name="sessionId">The session id of the client to
 subscribe.</param>
  /// <param name="topicSelector">The topic selector
 expression.</param>
  /// <param name="callback">The callback for the subscription
 result.</param>
  public void Subscribe( SessionId sessionId, string
 topicSelector, ISubscriptionCallback callback )
  {
   // To subscribe a client to a topic, this client session must
 have the MODIFY_SESSION permission.
   subscriptionControl.Subscribe( sessionId, topicSelector,
 callback );
  }
```

```
  /// <summary>
  /// Unsubscribe a client from topics.
  /// </summary>
  /// <param name="sessionId">The session id of the client to
 unsubscribe.</param>
  /// <param name="topicSelector">The topic selector
 expression.</param>
  /// <param name="callback">The callback for the unsubscription
 result.</param>
  public void Unsubscribe( SessionId sessionId, string
 topicSelector, ISubscriptionCallback callback )
  {
   subscriptionControl.Unsubscribe( sessionId, topicSelector,
 callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Example: Receive notifications when a client subscribes to a routing topic
The following examples use the SubscriptionControl feature in the Unified API to listen for notifications of when a client subscribes to a routing topic.

Java

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.SubscriptionCo
import
 com.pushtechnology.diffusion.client.features.control.topics.SubscriptionCo
import
 com.pushtechnology.diffusion.client.features.control.topics.SubscriptionCo
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This demonstrates using a control client to be notified of
 subscription
 * requests to routing topics.
 * <P>
 * This uses the 'SubscriptionControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientSubscriptionControlRouting {

    private final Session session;

    /**
```

```
     * Constructor.
     *
     * @param routingCallback for routing subscription requests
     */
    public ControlClientSubscriptionControlRouting(
        final SubscriptionCallback routingCallback) {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        final SubscriptionControl subscriptionControl =
            session.feature(SubscriptionControl.class);

        // Sets up a handler so that all subscriptions to topic
 a/b are routed
        // to routing/target/topic
        // To do this, the client session requires the
 'view_session',
        // 'modify_session', and 'register_handler' permissions.
        subscriptionControl.addRoutingSubscriptionHandler(
            "a/b",
            new
 SubscriptionControl.RoutingSubscriptionRequest.Handler
            .Default() {
                @Override
                public void onSubscriptionRequest(
                    final RoutingSubscriptionRequest request) {

                    request.route(
                        "routing/target/topic",
                        routingCallback);
                }

            });
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
}
```

.NET

```
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates using a control client to be notified of
 subscription requests to routing topics.
 ///
 /// This uses the <see cref="ISubscriptionControl"/> feature.
 /// </summary>
 public class ControlClientSubscriptionControlRouting
```

```
{
 #region Fields

 private readonly ISession session;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 /// <param name="routingCallback">The callback for routing
subscription requests.</param>
 public
ControlClientSubscriptionControlRouting( ISubscriptionCallback
routingCallback )
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  var subscriptionControl =
session.GetSubscriptionControlFeature();

  // Sets up a handler so that all subscriptions to topic 'a/b'
are routed to the routing/target topic.
  // To do this, the client session requires the VIEW_SESSION,
MODIFY_SESSION and REGISTER_HANDLER
  // permissions.
  subscriptionControl.AddRoutingSubscriptionHandler( "a/b", new
SubscriptionHandler( routingCallback ) );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Close the session.
 /// </summary>
 public void Close()
 {
  session.Close();
 }

 #endregion Public Methods

 #region Private Classes

 private class SubscriptionHandler :
RoutingSubscriptionRequestHandlerDefault
 {
  #region Fields

  private readonly ISubscriptionCallback theRoutingCallback;

  #endregion Fields

  #region Constructor

  public SubscriptionHandler( ISubscriptionCallback callback )
  {
```

```
    theRoutingCallback = callback;
   }

   #endregion Constructor

   #region Overrides

   /// <summary>
   /// A request to subscribe to a specific routing topic.
   /// </summary>
   /// <param name="request"></param>
   public override void
 OnSubscriptionRequest( IRoutingSubscriptionRequest request )
   {
    request.Route( "routing/target/topic", theRoutingCallback );
   }

   #endregion Overrides
  }

  #endregion Private Classes
 }
}
```

Change the URL from that provided in the example to the URL of the Diffusion server.

Java Unified API

The Java Unified API is bundled in a library called `diffusion-client.jar`.
The Java Unified API comprises a number of classes subordinate to the `com.pushtechnology.diffusion.client` package.

A JAR file that contains the Diffusion client library is available on the Push Technology Maven™ repository. You can access the client library JAR file by declaring the following dependency:

```
<dependency>
    <groupId>com.pushtechnology.diffusion</groupId>
    <artifactId>diffusion-client</artifactId>
    <version>version</version>
</dependency>
```

A JAR file that contains only the interfaces required to develop your client is also available on the Push Technology Maven repository. You can access the interfaces JAR file by declaring the following dependency:

```
<dependency>
    <groupId>com.pushtechnology.diffusion</groupId>
    <artifactId>diffusion-api</artifactId>
    <version>version</version>
</dependency>
```

For more information about setting up the Push Technology Maven repository, see Getting started with Java on page 353.

The Java library is provided with the Diffusion server installation in the `clients/java` folder.

Support

Table 49: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
Java	8 (recommended), 7 (supported) **Note:** We recommend that you run your clients on the JDK rather than the JRE.	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)

Features are sets of capabilities provided by the API.

Table 50: Capabilities provided by the API

Feature set	Supported capabilities
Authentication Control	Register an authentication handler that authenticates client connections.
Client Control	Receive notifications when client queue events occur Receive notifications when client sessions open Receive notifications when client sessions close Get session details for a client session Manage client queue conflation Manage client queue throttling
Messaging	Send messages through a topic path to the message handler of that topic path Receive messages sent through a topic path
Messaging Control	Send messages to a specific client or set of clients Receive messages sent to the message handler of a topic path
Pings	Ping the Diffusion server
Security	Change the security principal and credentials associated with the client session

Feature set	Supported capabilities
Subscription Control	Subscribe other clients to topics Handle routing topic requests
System Authentication Control	Define users that are authenticated by the system authentication handler.
Topic Control	Create topics Delete topics
Topics	Subscribe to topics Receive streaming updates on subscribed topics Fetch the current state of a topic Unsubscribe from topics
Topic Update Control	Register as a topic update source Update topics exclusively Update topics non-exclusively

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 51: Capabilities provided by the library

Library capability	Supported
Connect through load balancer	✓
Reconnect	✓
Connect through HTTP proxy	✓
Flow control	✓

Getting started with Java

Create a Java client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a topic to the console when the topic is updated.

Before you begin

To complete this example, you need a Diffusion server.

You also require either a named user that has a role with the read_topic permission or that anonymous client connections are assigned a role with the read_topic permission. For example, the "CLIENT" role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Include the client jar file on the build classpath of your Java client. You can use one of the following methods:
   - You can use Maven to declare the dependency. First add the Push Technology public repository to your `pom.xml` file:

     ```
     <repositories>
         <repository>
             <id>push-repository</id>
             <url>https://download.pushtechnology.com/maven/</
     url>
         </repository>
     </repositories>
     ```

     Next declare the following dependency in your `pom.xml` file:

     ```
     <dependency>
         <groupId>com.pushtechnology.diffusion</groupId>
         <artifactId>diffusion-client</artifactId>
         <version>version</version>
     </dependency>
     ```

     Where *version* is the Diffusion version, for example 5.5.3.
   - If you are not using Maven, you can include the `diffusion-client-`*`version`*`.jar` file that is located in the `clients/java` directory of your Diffusion server installation.
2. Create a client class that imports the following packages and classes:

   ```
   import com.pushtechnology.diffusion.client.Diffusion;
   import com.pushtechnology.diffusion.client.content.Content;
   import com.pushtechnology.diffusion.client.features.Topics;
   import
    com.pushtechnology.diffusion.client.features.Topics.TopicStream;
   import com.pushtechnology.diffusion.client.session.Session;
   import
    com.pushtechnology.diffusion.client.types.UpdateContext;

   public class SubscribingClient {

   }
   ```

3. Create a `main` method for the client.

   ```
   public class SubscribingClient {
       public static void main(String ... arguments) throws
    Exception {

       }
   }
   ```

4. In the `main` method, connect to the Diffusion server.

```
        // Connect anonymously
        // Replace 'host' with your hostname
        Session session =
Diffusion.sessions().open("ws://host:port");
```

Or you can connect securely, using :

```
        Session session =
Diffusion.sessions().open("ws://host:port");
```

Or you can connect with a principal and credentials if that principal is assigned a role with the read_topic permission:

```
Session session = Diffusion.sessions().principal("principal")
    .password("password").open("ws://host:port");
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

5. Next, in the `main` method, get the Topics feature.

```
        // Get the Topics feature to subscribe to topics
        final Topics topics = session.feature(Topics.class);
```

The Topics feature enables a client to subscribe to a topic or fetch its state. For more information, see Topics on page 185.

6. Within the `SubscribingClient` class, create an inner class that extends `TopicStream.Default` and overrides the `onTopicUpdate` method.

This inner class defines the behavior that occurs when a topic that the client subscribes to is updated. In this example, the topic stream prints the topic name and the content of the update to the console.

```
   private static class TopicStreamPrintLn extends
TopicStream.Default {
        @Override
        public void onTopicUpdate(String topic, Content
content,
                UpdateContext context) {

            System.out.println(topic + ":   " +
content.asString());
        }
```

7. Back in the `main` method of the `SubscribingClient` class, use the `addTopicStream` method to associate an instance of the topic stream that you created with the topic you want to subscribe to.

```
        // Add a new topic stream for 'foo/counter'
        topics.addTopicStream(">foo/counter", new
TopicStreamPrintLn());
```

8. Next, use the `subscribe` method to subscribe to the topic foo/counter.

```
        // Subscribe to the topic 'foo/counter'
        topics.subscribe("foo/counter",
                new Topics.CompletionCallback.Default());
```

9. Use a `Thread.sleep()` to hold the client open for a minute while the updates are received and output.

```
        // Wait for a minute while the stream prints updates
        Thread.sleep(60000);
```

10. Compile and run your client.

    We recommend that you run your client using the JDK rather than the JRE. The JDK includes additional diagnostic capabilities that might be useful.

Results

The client outputs the value to the console every time the value of the foo/counter topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. For more information, see Start publishing with Java on page 357.

Full example

The completed `SubscribingClient` class contains the following code:

```
import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.content.Content;
import com.pushtechnology.diffusion.client.features.Topics;
import
 com.pushtechnology.diffusion.client.features.Topics.TopicStream;
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.types.UpdateContext;

public class SubscribingClient {
    public static void main(String... arguments) throws
 Exception {

        // Connect anonymously
        // Replace 'host' with your hostname
        Session session =
 Diffusion.sessions().open("ws://host:port");

        // Get the Topics feature to subscribe to topics
        final Topics topics = session.feature(Topics.class);

        // Add a new topic stream for 'foo/counter'
        topics.addTopicStream(">foo/counter", new
 TopicStreamPrintLn());

        // Subscribe to the topic 'foo/counter'
        topics.subscribe("foo/counter",
                new Topics.CompletionCallback.Default());

        // Wait for a minute while the stream prints updates
        Thread.sleep(60000);
    }

    /**
     * A topic stream that prints updates to the console.
     */
    private static class TopicStreamPrintLn extends
 TopicStream.Default {
        @Override
        public void onTopicUpdate(String topic, Content
 content,
```

```
                UpdateContext context) {

            System.out.println(topic + ":   " +
 content.asString());
        }
    }
}
```

Start publishing with Java

Create a Java client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with Java installed on it.

You also require either a named user that has a role with the modify_topic and update_topic permissions. For example, the "ADMINISTRATOR" role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to publish a single value to a topic. The full code example is provided after the steps.

Procedure

1. Include the client jar file on the build classpath of your Java client. You can use one of the following methods:
   - You can use Maven to declare the dependency. First add the Push Technology public repository to your `pom.xml` file:

     ```
     <repositories>
         <repository>
             <id>push-repository</id>
             <url>https://download.pushtechnology.com/maven/</
     url>
         </repository>
     </repositories>
     ```

     Next declare the following dependency in your `pom.xml` file:

     ```
     <dependency>
         <groupId>com.pushtechnology.diffusion</groupId>
         <artifactId>diffusion-client</artifactId>
         <version>version</version>
     </dependency>
     ```

     Where *version* is the Diffusion version, for example 5.5.3.
   - If you are not using Maven, you can include the `diffusion-client.jar` file that is located in the `clients/java` directory of your Diffusion server installation.
2. Create a `PublishingClient` class that imports the following packages and classes:

   ```
   import com.pushtechnology.diffusion.client.Diffusion;
   import
    com.pushtechnology.diffusion.client.features.control.topics.TopicControl
   ```

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateC
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateC
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

import java.util.concurrent.CountDownLatch;

public final class PublishingClient {

}
```

The `com.pushtechnology.diffusion.client` packages contain the classes to use to interact with the Diffusion server. The `java.util.concurrent.CountDownLatch` class is used to simplify this example by making it more synchronous. However, the Diffusion Java API is designed to be most powerful when used asynchronously.

3. Create a `main` method.

```
public final class PublishingClient {

    public static void main(String... arguments) throws
 InterruptedException {

    }
}
```

4. Inside the `main` method, connect to the Diffusion server.

```
        // Connect using a principal with 'modify_topic' and
 'update_topic'
        // permissions
        Session session =
 Diffusion.sessions().principal("principal")
          .password("password").open("ws://host:port");
```

Or you can connect securely to the Diffusion server using :

```
.open("wss://host:port");
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

You can choose to connect anonymously if anonymous sessions are assigned the modify_topic and update_topic permissions. However, we do not recommend that anonymous sessions are given write access to data on the Diffusion server.

5. Next, in the `main` method, get the TopicControl and TopicUpdateControl features.

```
// Get the TopicControl and TopicUpdateControl feature
        TopicControl topicControl =
 session.feature(TopicControl.class);

        TopicUpdateControl updateControl = session
                .feature(TopicUpdateControl.class);
```

The TopicControl feature enables a client to create and delete topics. For more information, see TopicControl on page 219.

The TopicUpdateControl feature enables a client to publish updates to a topic. For more information, see TopicUpdateControl on page 259.

6. Next, in the `main` method, use the TopicControl feature to create the foo/counter topic.

```
        final CountDownLatch waitForStart = new
CountDownLatch(1);

        // Create a single value topic 'foo/counter'
        topicControl.addTopic("foo/counter",
TopicType.SINGLE_VALUE,
                new AddCallback.Default() {
                    @Override
                    public void onTopicAdded(String topicPath)
{
                        waitForStart.countDown();
                    }
                });

        // Wait for the onTopicAdded() callback.
        waitForStart.await();
```

This example uses a `CountDownLatch` to wait until the topic is successfully added. This approach is used to simplify the example and is not recommended for production clients.

7. Next, in the `main` method, loop once a second updating the foo/counter topic with an incrementing count from 0 to 1000.

   Use the `updateControl.updater().update()` method to update a topic without locking that topic.

```
        // Update the topic
        for (int i = 0; i < 1000; ++i) {

            // Use the non-exclusive updater to update the
topic without locking
            // it
            updateControl.updater().update("foo/counter",
Integer.toString(i),
                    new UpdateCallback.Default());

            Thread.sleep(1000);
        }
```

8. Compile and run your client.

   We recommend that you run your client using the JDK rather than the JRE. The JDK includes additional diagnostic capabilities that might be useful.

Results

The client publishes a value to the foo/counter topic every second. You can subscribe to the foo/counter topic by creating a client to subscribe to the topic. For more information, see Getting started with Java on page 353.

Full example

The completed `PublishingClient` class contains the following code:

```
import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
```

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCo
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCo
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

import java.util.concurrent.CountDownLatch;

public final class PublishingClient {

    public static void main(String... arguments) throws
 InterruptedException {

        // Connect using a principal with 'modify_topic' and
 'update_topic'
        // permissions
        Session session =
 Diffusion.sessions().principal("principal")
                .password("password").open("ws://host:port");

        // Get the TopicControl and TopicUpdateControl
 feature
        TopicControl topicControl =
 session.feature(TopicControl.class);

        TopicUpdateControl updateControl = session
                .feature(TopicUpdateControl.class);

        final CountDownLatch waitForStart = new
 CountDownLatch(1);

        // Create a single value topic 'foo/counter'
        topicControl.addTopic("foo/counter",
 TopicType.SINGLE_VALUE,
                new AddCallback.Default() {
                    @Override
                    public void onTopicAdded(String
 topicPath) {
                        waitForStart.countDown();
                    }
                });

        // Wait for the onTopicAdded() callback.
        waitForStart.await();

        // Update the topic
        for (int i = 0; i < 1000; ++i) {

            // Use the non-exclusive updater to update the
 topic without locking
            // it
            updateControl.updater().update("foo/counter",
 Integer.toString(i),
                    new UpdateCallback.Default());

            Thread.sleep(1000);
        }
    }
```

```
}
```

Using the Java Unified API

Considerations and capabilities that are specific to the Java Unified API.

Certificates

Diffusion Java clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the , you must set up a trust store for the client and add the appropriate certificates to that trust store.

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use keytool to create a trust store for your client that includes this certificate.

   For more information, see https://docs.oracle.com/cd/E19509-01/820-3503/ggfka/index.html
3. Use system properties to add the trust store to your client.

   For example:

   ```
   System.setProperty("javax.net.ssl.trustStore",
    "truststore_name");
   ```

   Or at the command line:

   ```
   -Djavax.net.ssl.keyStore=path_to_truststore
   ```

Java examples

Examples that use the Java Unified API.

- Reconnection example on page 361
- AuthenticationControl example on page 363
- ClientControl example on page 365
- Messaging example on page 366
- MessagingControl example on page 368
- Pings example on page 370
- Security example on page 371
- SystemAuthenticationControl example on page 372
- TopicControl example on page 375
- Topics example on page 383
- TopicUpdateControl example on page 386

Reconnection example

The following example shows using the Java Unified API to define the client's reconnection behavior.

```
package com.pushtechnology.diffusion.examples;
```

```
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.session.Session.Listener;
import
 com.pushtechnology.diffusion.client.session.Session.State;
import
 com.pushtechnology.diffusion.client.session.reconnect.ReconnectionStrategy


/**
 * This example class demonstrates the ability to set a custom
 {@link ReconnectionStrategy}
 * when creating sessions.
 *
 * @author Push Technology Limited
 * @since 5.5
 */
public class ClientWithReconnectionStrategy {

    /**
     * Constructor.
     */
    public ClientWithReconnectionStrategy() {

        // Set the maximum amount of time we'll try and
 reconnect for to 10 minutes.
        final int maximumTimeoutDuration = 1000 * 60 * 10;

        // Set the maximum interval between reconnect attempts
 to 60 seconds.
        final long maximumAttemptInterval = 1000 * 60;

        // Create a new reconnection strategy that applies an
 exponential backoff
        final ReconnectionStrategy reconnectionStrategy = new
 ReconnectionStrategy() {
            private final ScheduledExecutorService scheduler =
 Executors.newScheduledThreadPool(1);
            private volatile int retries = 0;

            @Override
            public void performReconnection(final
 ReconnectionAttempt reconnection) {
                final long exponentialWaitTime =
                    Math.min((long) Math.pow(2,  retries++) *
 100L, maximumAttemptInterval);

                scheduler.schedule(new Runnable() {
                    @Override
                    public void run() {
                        reconnection.start();
                    }
                }, exponentialWaitTime, TimeUnit.MILLISECONDS);
            }
        };

        final Session session =
 Diffusion.sessions().reconnectionTimeout(maximumTimeoutDuration)
```

```
 .reconnectionStrategy(reconnectionStrategy)
                                                        .open("ws://
diffusion.example.com:80");
        session.addListener(new Listener() {
            @Override
            public void onSessionStateChanged(Session session,
 State oldState, State newState) {

                if (newState == State.RECOVERING_RECONNECT) {
                    // The session has been disconnected, and
 has entered recovery state. It is during this state that
                    // the reconnect strategy will be called
                }

                if (oldState == State.RECOVERING_RECONNECT) {
                    // The session has left recovery state. It
 may either be attempting to reconnect, or the attempt has
                    // been aborted; this will be reflected in
 the newState.
                }
            }
        });
    }
}
```

AuthenticationControl example

The following example shows using the Java Unified API to register a control authentication handler with the Diffusion server. The example also includes a simple or empty authentication handler.

The name by which the control authentication handler is registered must be configured in the `Server.xml` configuration file of the Diffusion server for the control authentication handler to be called to handle authentication requests.

```
package com.pushtechnology.diffusion.examples;

import java.nio.charset.Charset;
import java.util.Arrays;
import java.util.EnumSet;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.details.SessionDetails;
import
 com.pushtechnology.diffusion.client.details.SessionDetails.DetailType;
import
 com.pushtechnology.diffusion.client.features.ServerHandler;
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticatio
import
 com.pushtechnology.diffusion.client.features.control.clients.Authenticatio
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.types.Credentials;

/**
 * This demonstrates the use of a control client to authenticate
 client
 * connections.
 * <P>
 * This uses the 'AuthenticationControl' feature.
```

```
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientIdentityChecks {

    private final Session session;

    /**
     * Constructor.
     */
    public ControlClientIdentityChecks() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        final AuthenticationControl authenticationControl =
            session.feature(AuthenticationControl.class);

        // To register the authentication handler, this client
 session must
        // have the 'authenticate' and 'register_handler'
 permissions.
        authenticationControl.setAuthenticationHandler(
            "example-handler",
            EnumSet.allOf(DetailType.class),
            new Handler());
    }

    /**
     * Authentication handler.
     */
    private static class Handler extends ServerHandler.Default
        implements ControlAuthenticationHandler {
        @Override
        public void authenticate(
            final String principal,
            final Credentials credentials,
            final SessionDetails sessionDetails,
            final Callback callback) {

            final byte[] passwordBytes =
                "password".getBytes(Charset.forName("UTF-8"));

            if ("admin".equals(principal) &&
                credentials.getType() ==
 Credentials.Type.PLAIN_PASSWORD &&
                Arrays.equals(credentials.toBytes(),
 passwordBytes)) {
                callback.allow();
            }
            else {
                callback.deny();
            }
        }
    }

    /**
     * Close the session.
     */
    public void close() {
```

```
        session.close();
    }
}
```

ClientControl example

The following example shows using the Java Unified API to set queue thresholds and register listeners for queue events.

```
package com.pushtechnology.diffusion.examples;

import static
 com.pushtechnology.diffusion.client.features.control.clients.MessageQueueP

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.ClientControl
import
 com.pushtechnology.diffusion.client.features.control.clients.MessageQueueP
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;

/**
 * This demonstrates the use of a control client to apply both
 throttling and
 * conflation to clients. It throttles and conflates all clients
 that reach
 * their queue thresholds and remove when they go down again.
 * <P>
 * This uses the 'ClientControl' feature.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientConflateAndThrottle {

    private final Session session;
    private final ClientControl clientControl;
    private final ClientCallback clientCallback;

    /**
     * Constructor.
     *
     * @param callback notifies callback from throttle requests
     */
    public ControlClientConflateAndThrottle(ClientCallback
 callback) {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        // Create the ClientControl feature with a handler that
 sets queue
        // thresholds on new connecting clients and sets a
 listener for queue
        // events.
```

```
        clientControl = session.feature(ClientControl.class);
        clientCallback = callback;

        // To register a queue event handler, the client session
 must have
        // the 'register handler' and 'view_session'
 permissions.
        clientControl.setQueueEventHandler(new
 MyThresholdHandler());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

    private class MyThresholdHandler extends
 QueueEventHandler.Default {

        @Override
        public void onUpperThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The setThrottled method enables throttling and
 conflation.
            // This method requires the client session to have
 the
            // 'modify_session' permission.
            clientControl.setThrottled(client, MESSAGE_INTERVAL,
 10,
                clientCallback);
        }

        @Override
        public void onLowerThresholdCrossed(
            final SessionId client,
            final MessageQueuePolicy policy) {

            // The setThrottled method enables throttling and
 conflation.
            // This method requires the client session to have
 the
            // 'modify_session' permission.
            clientControl
                .setThrottled(client, MESSAGE_INTERVAL, 1000,
 clientCallback);
        }
    }
}
```

Messaging example

The following example shows using the Java Unified API to send a message to a topic path.

```
package com.pushtechnology.diffusion.examples;

import java.util.List;

import com.pushtechnology.diffusion.client.Diffusion;
```

```
import com.pushtechnology.diffusion.client.features.Messaging;
import
 com.pushtechnology.diffusion.client.features.Messaging.SendCallback;
import
 com.pushtechnology.diffusion.client.features.Messaging.SendContextCallback
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This is a simple example of a client that uses the
 'Messaging' feature to
 * send messages to a topic path.
 * <P>
 * To send a message on a topic path, the client session
 requires the
 * 'send_to_message_handler' permission.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientSendingMessages {

    private final Session session;
    private final Messaging messaging;

    /**
     * Constructs a message sending application.
     */
    public ClientSendingMessages() {
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        messaging = session.feature(Messaging.class);
    }

    /**
     * Sends a simple string message to a specified topic path.
     * <P>
     * There will be no context with the message so callback
 will be directed to
     * the no context callback.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param callback notifies message sent
     */
    public void send(String topicPath, String message,
 SendCallback callback) {
        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),
            callback);
    }

    /**
     * Sends a simple string message to a specified topic path
 with context string.
     * <P>
     * Callback will be directed to the contextual callback with
 the string
     * provided.
     *
     * @param topicPath the topic path
```

```
     * @param message the message to send
     * @param context the context string to return with the
 callback
     * @param callback notifies message sent
     */
    public void send(
        String topicPath,
        String message,
        String context,
        SendContextCallback<String> callback) {

        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),
            context,
            callback);
    }

    /**
     * Sends a string message to a specified topic path with
 headers.
     * <P>
     * There will be no context with the message so callback
 will be directed to
     * the no context callback.
     *
     * @param topicPath the topic path
     * @param message the message to send
     * @param headers the headers to send with the message
     * @param callback notifies message sent
     */
    public void sendWithHeaders(
        String topicPath,
        String message,
        List<String> headers,
        SendCallback callback) {

        messaging.send(
            topicPath,
            Diffusion.content().newContent(message),

 messaging.sendOptionsBuilder().headers(headers).build(),
            callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

MessagingControl example

The following example shows using the Java Unified API to receive messages sent to topic paths and to send messages to one or more clients

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
```

```
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import
 com.pushtechnology.diffusion.client.features.control.topics.MessagingContr
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.session.SessionId;
import com.pushtechnology.diffusion.client.types.ReceiveContext;

/**
 * This is an example of a control client using the
 'MessagingControl' feature
 * to receive messages from clients and also send messages to
 clients.
 * <P>
 * It is a trivial example that simply responds to all messages
 on a particular
 * branch of the topic tree by echoing them back to the client
 exactly as they
 * are complete with headers.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientReceivingMessages {

    private final Session session;
    private final MessagingControl messagingControl;
    private final SendCallback sendCallback;

    /**
     * Constructor.
     *
     * @param callback for result of sends
     */
    public ControlClientReceivingMessages(SendCallback callback)
 {

        sendCallback = callback;

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");
        messagingControl =
 session.feature(MessagingControl.class);

        // Register to receive all messages sent by clients on
 the "foo" branch
        // To do this, the client session must have the
 'register_handler' permission.
        messagingControl.addMessageHandler("foo", new
 EchoHandler());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }
```

```
    /**
     * Handler that echoes messages back to the originating
 client complete with
     * original headers.
     */
    private class EchoHandler extends MessageHandler.Default {
        @Override
        public void onMessage(
            SessionId sessionId,
            String topicPath,
            Content content,
            ReceiveContext context) {

            // To send a message to a client, this client
 session must have
            // the 'view_session' and 'send_to_session'
 permissions.
            messagingControl.send(
                sessionId,
                topicPath,
                content,
                messagingControl.sendOptionsBuilder()
                    .headers(context.getHeaderList())
                    .build(),
                sendCallback);

        }
    }

}
```

Pings example

The following example shows using the Java Unified API to ping Diffusion from a client.

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Pings;
import
 com.pushtechnology.diffusion.client.features.Pings.PingContextCallback;
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This is a simple client example that pings the server and
 prints out the
 * round-trip time.
 * <P>
 * This uses the 'Pings' feature only.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientUsingPings {

    private final Session session;
    private final Pings pings;

    /**
     * Constructor.
     */
```

```
    public ClientUsingPings() {
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        pings = session.feature(Pings.class);
    }

    /**
     * Ping the server.
     *
     * @param context string to log with round trip time
     * @param callback used to return ping reply
     */
    public void ping(String context, PingContextCallback<String>
 callback) {
        pings.pingServer(context, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

Security example

The following example shows using the Java Unified API to change the client principal and credentials for a client session.

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Security;
import
 com.pushtechnology.diffusion.client.features.Security.ChangePrincipalCallb
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This demonstrates a client's use of credentials, specifically
 the ability to
 * change the principal for an active session.
 * <P>
 * This is not a realistic use case on its own, but is shown
 separately here for
 * clarity.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ClientUsingCredentials {

    private final Session session;
    private final Security security;

    /**
     * Constructor.
     */
    public ClientUsingCredentials() {
```

```
        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");
        security = session.feature(Security.class);
    }

    /**
     * Request a change of principal for the session.
     *
     * @param principal the new principal name
     * @param password the password
     * @param callback notifies success or failure
     */
    public void changePrincipal(
        String principal,
        String password,
        ChangePrincipalCallback callback) {

        security.changePrincipal(
            principal,
            Diffusion.credentials().password(password),
            callback);
    }

    /**
     * Close.
     */
    public void close() {
        session.close();
    }

}
```

SystemAuthenticationControl example

The following example shows using the Java Unified API to edit the contents of the system authentication store.

```
package com.pushtechnology.diffusion.examples;

import java.util.HashSet;
import java.util.Set;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.callbacks.ErrorReason;
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SystemAuthent
import
 com.pushtechnology.diffusion.client.features.control.clients.SecurityStore
```

```
import com.pushtechnology.diffusion.client.session.Session;

/**
 * An example of using a control client to alter the system
 authentication
 * configuration.
 * <P>
 * This uses the {@link SystemAuthenticationControl} feature
 only.
 *
 * @author Push Technology Limited
 * @since 5.2
 */
public class ControlClientChangingSystemAuthentication {

    private static final Logger LOG =
        LoggerFactory.getLogger(
            ControlClientChangingSystemAuthentication.class);

    private final SystemAuthenticationControl
 systemAuthenticationControl;

    /**
     * Constructor.
     */
    public ControlClientChangingSystemAuthentication() {

        final Session session = Diffusion.sessions()
            // Authenticate with a user that has the
 VIEW_SECURITY and
            // MODIFY_SECURITY permissions.
            .principal("admin").password("password")
            // Use a secure channel because we're transferring
 sensitive
            // information.
            .open("wss://diffusion.example.com:80");

        systemAuthenticationControl =
            session.feature(SystemAuthenticationControl.class);
    }

    /**
     * For all system users, update the assigned roles to
 replace the
     * "SUPERUSER" role and with "ADMINISTRATOR".
     *
     * @param callback result callback
     */
    public void
 changeSuperUsersToAdministrators(UpdateStoreCallback callback)
 {

        systemAuthenticationControl.getSystemAuthentication(
            new ChangeSuperUsersToAdministrators(callback));
    }

    private final class ChangeSuperUsersToAdministrators
        implements ConfigurationCallback {

        private final UpdateStoreCallback callback;

        ChangeSuperUsersToAdministrators(UpdateStoreCallback
 callback) {
```

```
            this.callback = callback;
        }

        @Override
        public void onReply(SystemAuthenticationConfiguration
 configuration) {

            ScriptBuilder builder =
                systemAuthenticationControl.scriptBuilder();

            // For all system users ...
            for (SystemPrincipal principal :
 configuration.getPrincipals()) {

                final Set<String> assignedRoles =
 principal.getAssignedRoles();

                // ... that have the SUPERUSER assigned role ...
                if (assignedRoles.contains("SUPERUSER")) {
                    final Set<String> newRoles = new
 HashSet<>(assignedRoles);
                    newRoles.remove("SUPERUSER");
                    newRoles.add("ADMINISTRATOR");

                    // ... add a command to the script that
 updates the user's
                    // assigned roles, replacing SUPERUSER with
 "ADMINISTRATOR".
                    builder =
                        builder.assignRoles(principal.getName(),
 newRoles);
                }
            }

            final String script = builder.script();

            LOG.info(
                "Sending the following script to the server:
\n{}",
                script);

            systemAuthenticationControl.updateStore(
                script,
                callback);
        }

        @Override
        public void onError(ErrorReason errorReason) {
            // This might fail if the session lacks the required
 permissions.
            callback.onError(errorReason);
        }
    }

    /**
     * Close the session.
     */
    public void close() {
        systemAuthenticationControl.getSession().close();
    }
}
```

TopicControl example

The following example shows using the Java Unified API to create topics.

```
package com.pushtechnology.diffusion.examples;

import java.util.List;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.callbacks.TopicTreeHandler;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.content.RecordContentBuilder;
import
 com.pushtechnology.diffusion.client.content.metadata.MContent;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.RecordTopicDetails;
import
 com.pushtechnology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client to add topics.
 * <P>
 * This uses the 'TopicControl' feature only.
 * <P>
 * To add or remove topics, the client session must have the
 'modify_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientAddingTopics {

    private final Session session;

    private final TopicControl topicControl;

    /**
     * Constructor.
     */
    public ControlClientAddingTopics() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);

    }

    /**
     * Adds a topic with type derived from value.
     * <P>
```

```
    * This uses the simple convenience method for adding topics
where the topic
    * type and metadata are derived from a supplied value which
can be any
    * object. For example, an Integer would result in a single
value topic of
    * type integer.
    *
    * @param topicPath full topic path
    * @param initialValue an optional initial value for the
topic
    * @param context this will be passed back to the callback
when reporting
    *        success or failure of the topic add
    * @param callback to notify result of operation
    * @param <T> the value type
    * @return the topic details used to add the topic
    */
   public <T> TopicDetails addTopicForValue(
       String topicPath,
       T initialValue,
       String context,
       AddContextCallback<String> callback) {

       return topicControl.addTopicFromValue(
           topicPath,
           initialValue,
           context,
           callback);
   }

   /**
    * Add a record topic from a list of initial values.
    * <P>
    * This demonstrates the simplest mechanism for adding a
record topic by
    * supplying values that both the metadata and the initial
values are
    * derived from.
    *
    * @param topicPath full topic path
    * @param initialValues the initial values for the topic
fields which will
    *        also be used to derive the metadata definition of
the topic
    * @param context this will be passed back to the callback
when reporting
    *        success or failure of the topic add
    * @param callback to notify result of operation
    * @return the topic details used to add the topic
    */
   public TopicDetails addRecordTopic(
       String topicPath,
       List<String> initialValues,
       String context,
       AddContextCallback<String> callback) {

       return topicControl.addTopicFromValue(
           topicPath,

Diffusion.content().newBuilder(RecordContentBuilder.class)
               .putFields(initialValues).build(),
           context,
```

```
            callback);

    }

    /**
     * Adds a record topic with supplied metadata and optional
initial content.
     * <P>
     * This example shows details being created and would be
fine when creating
     * topics that are all different but if creating many record
topics with the
     * same details then it is far more efficient to pre-create
the details.
     *
     * @param topicPath the full topic path
     * @param metadata pre-created record metadata
     * @param initialValue optional initial value for the topic
which must have
     *        been created to match the supplied metadata
     * @param context context passed back to callback when topic
created
     * @param callback to notify result of operation
     */
    public void addRecordTopic(
        String topicPath,
        MContent metadata,
        Content initialValue,
        String context,
        AddContextCallback<String> callback) {

        final TopicDetails details =

topicControl.newDetailsBuilder(RecordTopicDetails.Builder.class)
                .metadata(metadata).build();

        topicControl.addTopic(
            topicPath,
            details,
            initialValue,
            context,
            callback);
    }

    /**
     * Remove a single topic given its path.
     *
     * @param topicPath the topic path
     * @param callback notifies result of operation
     */
    public void removeTopic(String topicPath, RemoveCallback
callback) {
        topicControl.removeTopics(
            ">" + topicPath, // convert to a topic path selector
            callback);
    }

    /**
     * Remove one or more topics using a topic selector
expression.
     *
     * @param topicSelector the selector expression
     * @param callback notifies result of operation
```

```
     */
    public void removeTopics(String topicSelector,
 RemoveCallback callback) {
        topicControl.removeTopics(
            topicSelector,
            callback);
    }

    /**
     * Request that the topic {@code topicPath} and its
 descendants be removed
     * when the session is closed (either explicitly using
 {@link Session#close}
     * , or by the server). If more than one session calls this
 method for the
     * same {@code topicPath}, the topics will be removed when
 the last session
     * is closed.
     *
     * <p>
     * Different sessions may call this method for the same
 topic path, but not
     * for topic paths above or below existing registrations on
 the same branch
     * of the topic tree.
     *
     * @param topicPath the part of the topic tree to remove
 when the last
     *        session is closed
     */
    public void removeTopicsWithSession(String topicPath) {
        topicControl.removeTopicsWithSession(
            topicPath, new TopicTreeHandler.Default());
    }

    /**
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

The following example shows using the Java Unified API to create paged topics.

```
package com.pushtechnology.diffusion.examples;

import java.util.Collection;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.content.update.PagedRecordOrderedUpdat
import
 com.pushtechnology.diffusion.client.content.update.PagedStringUnorderedUpd
import
 com.pushtechnology.diffusion.client.content.update.Update;
```

```
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update
 paged topics.
 * <P>
 * This uses the 'TopicControl' feature to create a paged topic
 and the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * This demonstrates some simple examples of paged topic updates
 but not all of
 * the possible ways in which they can be done.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientUpdatingPagedTopics {

    private static final String ORDERED_TOPIC = "Paged/Ordered";
    private static final String UNORDERED_TOPIC = "Paged/
Unordered";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final PagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
    private final PagedStringUnorderedUpdateFactory
 unorderedUpdateFactory;
    private Updater pagedUpdater = null;

    /**
     * Constructor.
     */
    public ControlClientUpdatingPagedTopics() {

        session =

 Diffusion.sessions().principal("control").password("password")
```

```
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl =
session.feature(TopicUpdateControl.class);

        orderedUpdateFactory =
            updateControl

.updateFactory(PagedRecordOrderedUpdateFactory.class);
        unorderedUpdateFactory =
            updateControl

.updateFactory(PagedStringUnorderedUpdateFactory.class);

        final MetadataFactory metadata = Diffusion.metadata();

        // Create an unordered paged string topic
        topicControl.addTopic(
            UNORDERED_TOPIC,
            topicControl.newDetails(TopicType.PAGED_STRING),
            new AddCallback.Default());

        // Create an ordered paged record topic
        final MRecord recordMetadata =
            metadata.record(
                "Record",
                metadata.string("Name"),
                metadata.string("Address"));

        topicControl.addTopic(
            ORDERED_TOPIC,
            topicControl.

newDetailsBuilder(PagedRecordTopicDetails.Builder.class).
                metadata(recordMetadata).
                order(new OrderKey("Name")).build(),
            new AddCallback.Default());

        // Register as updater for topics under the 'Paged'
branch
        updateControl.registerUpdateSource(
            "Paged",
            new UpdateSource.Default() {
                @Override
                public void onActive(String topicPath, Updater
updater) {
                    pagedUpdater = updater;
                }
            });
    }

    /**
     * Add a new line to the ordered topic.
     *
     * @param name the name field value
     * @param address the address field value
     * @param callback to notify result
     */
    public void addOrdered(
        String name,
        String address,
        UpdateCallback callback) {
```

```
        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.add(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Update a line of an ordered topic.
     *
     * @param name the name of the line to update
     * @param address the new address field value
     * @param callback to notify result
     */
    public void updateOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.update(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Remove a line from an ordered topic.
     *
     * @param name the name of the line to remove
     * @param callback to notify result
     */
    public void removeOrdered(String name, UpdateCallback
callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.remove(
                Diffusion.content().newRecord(name, "")),
            callback);
    }

    /**
     * Add a line or lines to the end of an unordered topic.
     *
     * @param values lines to add
     * @param callback to notify result
     */
    public void addUnordered(
        Collection<String> values,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.add(values),
            callback);
    }

    /**
     * Insert a line or lines at a specified index within an
unordered topic.
     *
```

```
     * @param index the index to add at
     * @param values lines to insert
     * @param callback to notify result
     */
    public void insertUnordered(
        int index,
        Collection<String> values,
        UpdateCallback callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.insert(index, values),
            callback);
    }

    /**
     * Update a line within an unordered topic.
     *
     * @param index the index of the line to update
     * @param value the new line value
     * @param callback to notify result
     */
    public void updateUnordered(
        int index,
        String value,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.update(index, value),
            callback);
    }

    /**
     * Remove a specific line from an unordered topic.
     *
     * @param index the line to remove
     * @param callback to notify result
     */
    public void removeUnordered(int index, UpdateCallback
callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.remove(index),
            callback);
    }

    private void update(String topic, Update update,
UpdateCallback callback)
        throws IllegalStateException {
        if (pagedUpdater == null) {
            throw new IllegalStateException("No updater");
        }
        pagedUpdater.update(topic, update, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">Paged",
            new RemoveCallback() {
```

```
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
}
```

Topics example

The following example shows using the Java Unified API to subscribe to topics and receive the topic content.

```
package com.pushtechnology.diffusion.examples;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.content.Content;
import
 com.pushtechnology.diffusion.client.content.RecordContentReader;
import com.pushtechnology.diffusion.client.features.Topics;
import
 com.pushtechnology.diffusion.client.features.Topics.TopicStream;
import com.pushtechnology.diffusion.client.session.Session;
import com.pushtechnology.diffusion.client.types.UpdateContext;

/**
 * In this simple and commonest case for a client we just
 subscribe to a few
 * topics and assign handlers for each to receive content.
 * <P>
 * This makes use of the 'Topics' feature only.
 * <P>
 * To subscribe to a topic, the client session must have the
 'read_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientSimpleSubscriber {

    private static final Logger LOG =
        LoggerFactory.getLogger(ClientSimpleSubscriber.class);

    private final Session session;

    /**
     * Constructor.
     */
    public ClientSimpleSubscriber() {

        session =
```

```
Diffusion.sessions().principal("client").password("password")
              .open("ws://diffusion.example.com:80");

       // Use the Topics feature to add a topic stream for
       // Foo and all topics under Bar and request subscription
to those topics
       final Topics topics = session.feature(Topics.class);
       topics.addTopicStream(">Foo", new FooTopicStream());
       topics.addTopicStream(">Bar/", new BarTopicStream());
       topics.subscribe(
           Diffusion.topicSelectors().anyOf("Foo", "Bar//"),
           new Topics.CompletionCallback.Default());
   }

   /**
    * Close session.
    */
   public void close() {
       session.close();
   }

   /**
    * The topic stream for all messages on the 'Foo' topic.
    */
   private class FooTopicStream extends TopicStream.Default {
       @Override
       public void onTopicUpdate(
           String topic,
           Content content,
           UpdateContext context) {

           LOG.info(content.asString());
       }
   }

   /**
    * The topic stream for all messages on 'Bar' topics.
    */
   private class BarTopicStream extends TopicStream.Default {
       @Override
       public void onTopicUpdate(
           String topic,
           Content content,
           UpdateContext context) {

           // Process the message - one with a record with a
variable number of
           // fields followed by two more fields (effectively
another record
           // but no need to process as such).

           final RecordContentReader reader =
               Diffusion.content().newReader(
                   RecordContentReader.class,
                   content);

           for (String field : reader.nextRecord()) {
               LOG.info("Record 1 Field={}", field);
           }

           LOG.info("Extra Field 1={}", reader.nextField());
           LOG.info("Extra Field 2={}", reader.nextField());
```

```
        }
    }
}
```

The following example shows using the Java Unified API to fetch the current state of a topic without subscribing to the topic.

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import com.pushtechnology.diffusion.client.features.Topics;
import
 com.pushtechnology.diffusion.client.features.Topics.FetchContextStream;
import com.pushtechnology.diffusion.client.session.Session;

/**
 * This is a simple example of a client that fetches the state
 of topics but
 * does not subscribe to them.
 * <P>
 * This makes use of the 'Topics' feature only.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public final class ClientUsingFetch {

    private final Session session;
    private final Topics topics;

    /**
     * Constructor.
     */
    public ClientUsingFetch() {

        session =

 Diffusion.sessions().principal("client").password("password")
                .open("ws://diffusion.example.com:80");

        topics = session.feature(Topics.class);
    }

    /**
     * Issues a fetch request for a topic or selection of
 topics.
     *
     * @param topicSelector a {@link TopicSelector} expression
     * @param fetchContext context string to be returned with
 the fetch
     *        response(s)
     * @param stream callback for fetch responses
     */
    public void fetch(
        String topicSelector,
        String fetchContext,
        FetchContextStream<String> stream) {

        topics.fetch(topicSelector, fetchContext, stream);
    }

    /**
```

```
     * Close the session.
     */
    public void close() {
        session.close();
    }

}
```

TopicUpdateControl example

The following example shows using the Java Unified API to update topics exclusively.

```
package com.pushtechnology.diffusion.examples;

import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.SingleValueTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicDetails;

/**
 * An example of using a control client as an event feed to a
 topic.
 * <P>
 * This uses the 'TopicControl' feature to create a topic and
 the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientAsUpdateSource {

    private static final String TOPIC_NAME = "Feeder";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final UpdateCallback updateCallback;
```

```
    /**
     * Constructor.
     *
     * @param callback for updates
     */
    public ControlClientAsUpdateSource(UpdateCallback callback)
{

        updateCallback = callback;

        session =

Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");
        topicControl = session.feature(TopicControl.class);
        updateControl =
session.feature(TopicUpdateControl.class);
    }

    /**
     * Start the feed.
     *
     * @param provider the provider of prices
     * @param scheduler a scheduler service to schedule a
periodic feeder task
     */
    public void start(
        final PriceProvider provider,
        final ScheduledExecutorService scheduler) {

        // Set up topic details
        final SingleValueTopicDetails.Builder builder =
            topicControl

.newDetailsBuilder(SingleValueTopicDetails.Builder.class);
        final TopicDetails details =

builder.metadata(Diffusion.metadata().decimal("Price")).build();

        // Declare a custom update source implementation. When
the source is set
        // as active start a periodic task to poll the provider
every second and
        // update the topic. When the source is closed, stop the
scheduled task.
        final UpdateSource source = new UpdateSource.Default() {
            private ScheduledFuture<?> theFeeder;

            @Override
            public void onActive(String topicPath, Updater
updater) {
                theFeeder =
                    scheduler.scheduleAtFixedRate(
                        new FeederTask(provider, updater),
                        1, 1, TimeUnit.SECONDS);
            }

            @Override
            public void onClose(String topicPath) {
                if (theFeeder != null) {
                    theFeeder.cancel(true);
                }
            }
```

```
        };

        // Create the topic. When the callback indicates that
the topic has been
        // created then register the topic source for the topic.
        topicControl.addTopic(
            TOPIC_NAME,
            details,
            new AddCallback.Default() {
                @Override
                public void onTopicAdded(String topic) {
                    updateControl.registerUpdateSource(topic,
source);
                }
            });

    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">" + TOPIC_NAME,
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }

    /**
     * Periodic task to poll from provider and send update to
server.
     */
    private final class FeederTask implements Runnable {

        private final PriceProvider priceProvider;
        private final Updater priceUpdater;

        private FeederTask(PriceProvider provider, Updater
updater) {
            priceProvider = provider;
            priceUpdater = updater;
        }

        @Override
        public void run() {
            priceUpdater.update(
                TOPIC_NAME,

Diffusion.content().newContent(priceProvider.getPrice()),
                updateCallback);
        }
```

```
    }

    /**
     * Interface of a price provider that can periodically be
 polled for a
     * price.
     */
    public interface PriceProvider {
        /**
         * Get the current price.
         *
         * @return current price as a decimal string
         */
        String getPrice();
    }
}
```

The following example shows using the Java Unified API to send updates to topics non-exclusively.

```
package com.pushtechnology.diffusion.examples;

import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update a
 topic in non
 * exclusive mode (as opposed to acting as an exclusive update
 source). In this
 * mode other clients could update the same topic (on a last
 update wins basis).
 * <P>
 * This uses the 'TopicControl' feature to create a topic and
 the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * To send updates to a topic, the client session requires the
 'update_topic'
 * permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.3
 */
public final class ControlClientUpdatingTopic {

    private static final String TOPIC = "MyTopic";

    private final Session session;
    private final TopicControl topicControl;
```

```
    private final TopicUpdateControl updateControl;

    /**
     * Constructor.
     */
    public ControlClientUpdatingTopic() {

        session =

 Diffusion.sessions().principal("control").password("password")
                .open("ws://diffusion.example.com:80");

        topicControl = session.feature(TopicControl.class);
        updateControl =
 session.feature(TopicUpdateControl.class);

        // Create a single value topic
        topicControl.addTopic(
            TOPIC,
            TopicType.SINGLE_VALUE,
            new AddCallback.Default());

    }

    /**
     * Update the topic with a string value.
     *
     * @param value the update value
     * @param callback the update callback
     */
    public void update(String value, UpdateCallback callback) {
        updateControl.updater().update(TOPIC, value, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">" + TOPIC,
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
}
```

The following example shows using the Java Unified API to update paged topics.

```
package com.pushtechnology.diffusion.examples;

import java.util.Collection;
```

```
import com.pushtechnology.diffusion.client.Diffusion;
import
 com.pushtechnology.diffusion.client.content.metadata.MRecord;
import
 com.pushtechnology.diffusion.client.content.metadata.MetadataFactory;
import
 com.pushtechnology.diffusion.client.content.update.PagedRecordOrderedUpdat
import
 com.pushtechnology.diffusion.client.content.update.PagedStringUnorderedUpd
import
 com.pushtechnology.diffusion.client.content.update.Update;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl;
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.A
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicControl.R
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import
 com.pushtechnology.diffusion.client.features.control.topics.TopicUpdateCon
import com.pushtechnology.diffusion.client.session.Session;
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.PagedRecordTopicDetails
import
 com.pushtechnology.diffusion.client.topics.details.TopicType;

/**
 * An example of using a control client to create and update
 paged topics.
 * <P>
 * This uses the 'TopicControl' feature to create a paged topic
 and the
 * 'TopicUpdateControl' feature to send updates to it.
 * <P>
 * This demonstrates some simple examples of paged topic updates
 but not all of
 * the possible ways in which they can be done.
 * <P>
 * To send updates to a topic, the client session requires the
 * 'update_topic' permission for that branch of the topic tree.
 *
 * @author Push Technology Limited
 * @since 5.0
 */
public class ControlClientUpdatingPagedTopics {

    private static final String ORDERED_TOPIC = "Paged/Ordered";
    private static final String UNORDERED_TOPIC = "Paged/
Unordered";

    private final Session session;
    private final TopicControl topicControl;
    private final TopicUpdateControl updateControl;
    private final PagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
```

```
   private final PagedStringUnorderedUpdateFactory
unorderedUpdateFactory;
   private Updater pagedUpdater = null;

   /**
    * Constructor.
    */
   public ControlClientUpdatingPagedTopics() {

       session =

Diffusion.sessions().principal("control").password("password")
               .open("ws://diffusion.example.com:80");

       topicControl = session.feature(TopicControl.class);
       updateControl =
session.feature(TopicUpdateControl.class);

       orderedUpdateFactory =
           updateControl

.updateFactory(PagedRecordOrderedUpdateFactory.class);
       unorderedUpdateFactory =
           updateControl

.updateFactory(PagedStringUnorderedUpdateFactory.class);

       final MetadataFactory metadata = Diffusion.metadata();

       // Create an unordered paged string topic
       topicControl.addTopic(
           UNORDERED_TOPIC,
           topicControl.newDetails(TopicType.PAGED_STRING),
           new AddCallback.Default());

       // Create an ordered paged record topic
       final MRecord recordMetadata =
           metadata.record(
               "Record",
               metadata.string("Name"),
               metadata.string("Address"));

       topicControl.addTopic(
           ORDERED_TOPIC,
           topicControl.

newDetailsBuilder(PagedRecordTopicDetails.Builder.class).
               metadata(recordMetadata).
               order(new OrderKey("Name")).build(),
           new AddCallback.Default());

       // Register as updater for topics under the 'Paged'
branch
       updateControl.registerUpdateSource(
           "Paged",
           new UpdateSource.Default() {
               @Override
               public void onActive(String topicPath, Updater
updater) {
                   pagedUpdater = updater;
               }
           });
   }
```

```
    /**
     * Add a new line to the ordered topic.
     *
     * @param name the name field value
     * @param address the address field value
     * @param callback to notify result
     */
    public void addOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.add(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Update a line of an ordered topic.
     *
     * @param name the name of the line to update
     * @param address the new address field value
     * @param callback to notify result
     */
    public void updateOrdered(
        String name,
        String address,
        UpdateCallback callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.update(
                Diffusion.content().newRecord(name, address)),
            callback);
    }

    /**
     * Remove a line from an ordered topic.
     *
     * @param name the name of the line to remove
     * @param callback to notify result
     */
    public void removeOrdered(String name, UpdateCallback
callback) {

        update(
            ORDERED_TOPIC,
            orderedUpdateFactory.remove(
                Diffusion.content().newRecord(name, "")),
            callback);
    }

    /**
     * Add a line or lines to the end of an unordered topic.
     *
     * @param values lines to add
     * @param callback to notify result
     */
    public void addUnordered(
        Collection<String> values,
```

```
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.add(values),
            callback);
    }

    /**
     * Insert a line or lines at a specified index within an
unordered topic.
     *
     * @param index the index to add at
     * @param values lines to insert
     * @param callback to notify result
     */
    public void insertUnordered(
        int index,
        Collection<String> values,
        UpdateCallback callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.insert(index, values),
            callback);
    }

    /**
     * Update a line within an unordered topic.
     *
     * @param index the index of the line to update
     * @param value the new line value
     * @param callback to notify result
     */
    public void updateUnordered(
        int index,
        String value,
        UpdateCallback callback) {

        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.update(index, value),
            callback);
    }

    /**
     * Remove a specific line from an unordered topic.
     *
     * @param index the line to remove
     * @param callback to notify result
     */
    public void removeUnordered(int index, UpdateCallback
callback) {
        update(
            UNORDERED_TOPIC,
            unorderedUpdateFactory.remove(index),
            callback);
    }

    private void update(String topic, Update update,
UpdateCallback callback)
        throws IllegalStateException {
        if (pagedUpdater == null) {
            throw new IllegalStateException("No updater");
```

```
        }
        pagedUpdater.update(topic, update, callback);
    }

    /**
     * Close the session.
     */
    public void close() {
        // Remove our topic and close session when done
        topicControl.removeTopics(
            ">Paged",
            new RemoveCallback() {
                @Override
                public void onDiscard() {
                    session.close();
                }

                @Override
                public void onTopicsRemoved() {
                    session.close();
                }
            });

    }
}
```

.NET Unified API

The .NET Unified API is provided in the `PushTechnology.ClientInterface.dll` file.

The .NET assembly is provided with the Diffusion server installation in the `clients/dotnet` folder.

Support

Table 52: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
.NET	4.5	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)

Features are sets of capabilities provided by the API.

Table 53: Capabilities provided by the API

Feature set	Supported capabilities
Authentication Control	Register an authentication handler that authenticates client connections.
Client Control	Receive notifications when client queue events occur Receive notifications when client sessions open

Feature set	Supported capabilities
	Receive notifications when client sessions close Get session details for a client session Manage client queue conflation Manage client queue throttling
Messaging	Send messages through a topic path to the message handler of that topic path Receive messages sent through a topic path
Messaging Control	Send messages to a specific client or set of clients Receive messages sent to the message handler of a topic path
Pings	Ping the Diffusion server
Security	Change the security principal and credentials associated with the client session
Subscription Control	Subscribe other clients to topics Handle routing topic requests
System Authentication Control	Define users that are authenticated by the system authentication handler.
Topic Control	Create topics Delete topics
Topics	Subscribe to topics Receive streaming updates on subscribed topics Fetch the current state of a topic Unsubscribe from topics
Topic Update Control	Register as a topic update source Update topics exclusively Update topics non-exclusively

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 54: Capabilities provided by the library

Library capability	Supported
Connect through load balancer	✓
Reconnect	✓
Connect through HTTP proxy	✓
Flow control	✓

Getting started with .NET

Create a .NET client within minutes that connects to the Diffusion server. This example creates a client that prints the value of a topic to the console when the topic is updated.

Before you begin

To complete this example, you need a Diffusion server.

You also require either a named user that has a role with the read_topic permission or that anonymous client connections are assigned a role with the read_topic permission. For example, the "CLIENT" role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Create a .NET project that references the following DLL file located in the `clients/dotnet` directory of your .NET installation:

   `PushTechnology.ClientInterface.dll`

   The assembly contains the interfaces classes and interfaces that you use when creating a control client.

2. In your project, create a C# file that uses the following packages:

```
using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace Examples {
    class Program {

    }
}
```

3. Create a `Main` method.

```
    class Program
    {
        static void Main( string[] args ) {

        }
    }
}
```

4. In the `Main` method, connect to the Diffusion server.

```
        static void Main( string[] args ) {
            // Connect anonymously
            // Replace 'host' with your hostname
            var session =
Diffusion.Sessions.Open("ws://host:port");
        }
```

Or you can connect securely, using :

```
        Session session =
Diffusion.sessions().Open("ws://host:port");
```

Or you can connect with a principal and credentials if that principal is assigned a role with the read_topic permission:

```
Session session = Diffusion.Sessions().Principal("principal")
    .Password("password").Open("ws://host:port");
```

Replace the *host*, *port*, *principal*, and *password* values with your own information.

5. Next, in the `Main` method, get the Topics feature.

```
            // Get the Topics feature to subscribe to topics
            var topics = session.GetTopicsFeature();
```

The Topics feature enables a client to subscribe to a topic or fetch its state. For more information, see Topics on page 185.

6. Within the subscribing client class, create an inner class that implements `ITopicStream` and overrides the `OnTopicUpdate` method.

This inner class defines the behavior that occurs when a topic that the client subscribes to is updated. In this example, the topic stream prints the topic name and the content of the update to the console.

```
        /**
        * A topic stream that prints updates to the console.
        */
        private class TopicStreamPrintLn : ITopicStream {
            public void OnSubscription( string topicPath,
ITopicDetails details ) {
            }

            public void OnUnsubscription( string topicPath,
TopicUnsubscribeReason reason ) {
            }

            public void OnTopicUpdate( string topicPath,
IContent content, IUpdateContext context )
```

```
            {
                Console.WriteLine(topicPath + ":   " +
content);
            }

            public void OnClose() {
            }

            public void OnError( ErrorReason errorReason ) {
            }
        }
```

7. Back in the `Main` method of the subscribing client class, use the `AddTopicStream` method to associate an instance of the topic stream that you created with the topic you want to subscribe to.

```
            // Add a new topic stream for 'foo/counter'
            topics.AddTopicStream(">foo/counter", new
TopicStreamPrintLn());
```

8. Next, use the `Subscribe` method to subscribe to the topic.

   In this example, the topic subscribed to is foo/counter.

```
            // Subscribe to the topic 'foo/counter'
            topics.Subscribe("foo/counter", new
TopicsCompletionCallbackDefault());
```

9. Use a `Thread.sleep()` to hold the client open for a minute while the updates are received and output.

```
            // Wait for a minute while the stream prints
updates
            Thread.Sleep(60000);
```

10. Compile and run your client.

Results

The client outputs the value to the console every time the value of the foo/counter topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. For more information, see Start publishing with .NET on page 400.

Full example

The completed subscribing client class contains the following code:

```
using System;
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace Examples {
    class Program {
        /**
        * A topic stream that prints updates to the console.
        */
        private class TopicStreamPrintLn : ITopicStream {
```

```
            public void OnSubscription( string topicPath,
 ITopicDetails details ) {
            }

            public void OnUnsubscription( string topicPath,
 TopicUnsubscribeReason reason ) {
            }

            public void OnTopicUpdate( string topicPath,
 IContent content, IUpdateContext context )
            {
                Console.WriteLine(topicPath + ":   " +
 content);
            }

            public void OnClose() {
            }

            public void OnError( ErrorReason errorReason ) {
            }
        }

        static void Main( string[] args ) {
            // Connect anonymously
            // Replace 'host' with your hostname
            var session =
 Diffusion.Sessions.Open("ws://host:port");

            // Get the Topics feature to subscribe to topics
            var topics = session.GetTopicsFeature();

            // Add a new topic stream for 'foo/counter'
            topics.AddTopicStream(">foo/counter", new
 TopicStreamPrintLn());

            // Subscribe to the topic 'foo/counter'
            topics.Subscribe("foo/counter", new
 TopicsCompletionCallbackDefault());

            // Wait for a minute while the stream prints
 updates
            Thread.Sleep(60000);
        }
    }
}
```

Start publishing with .NET

Create a .NET client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with the .NET Framework installed on it.

You also require either a named user that has a role with the modify_topic and update_topic permissions. For example, the “ADMINISTRATOR” role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Create a .NET project that references the following DLL file located in the `clients/dotnet` directory of your .NET installation:

   `PushTechnology.ClientInterface.dll`

   The assembly contains the interfaces classes and interfaces that you use when creating a control client.

2. In your project, create a C# file that uses the following packages:

```
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Topics;

namespace Examples {
    class Program
    {

    }
}
```

3. Create a `Main` method.

```
    class Program
    {
        static void Main( string[] args ) {

        }
    }
}
```

4. Inside the `Main` method, connect to the Diffusion server.

```
        static void Main( string[] args ) {
            // Connect using a principal with 'modify_topic'
 and 'update_topic'
            // permissions
            var session =
 Diffusion.Sessions.Principal("principal").Password("password").Open("ws:
        }
```

   Or you can connect securely to the Diffusion server using :

```
.Open("wss://host:port");
```

   Replace the *host*, *port*, *principal*, and *password* values with your own information.

   You can choose to connect anonymously if anonymous sessions are assigned the modify_topic and update_topic permissions. However, we do not recommend that anonymous sessions are given write access to data on the Diffusion server.

5. Next, in the `Main` method, get the TopicControl and TopicUpdateControl features.

```
            // Get the TopicControl and TopicUpdateControl
feature
            var topicControl =
session.GetTopicControlFeature();
            var updateControl =
session.GetTopicUpdateControlFeature();
```

The TopicControl feature enables a client to create and delete topics. For more information, see TopicControl on page 219.

The TopicUpdateControl feature enables a client to publish updates to a topic. For more information, see TopicUpdateControl on page 259.

6. Next, in the `Main` method, use the TopicControl feature to create the foo/counter topic.

```
            // Create a single value topic 'foo/counter'
            topicControl.AddTopic("foo/counter",
TopicType.SINGLE_VALUE, new TopicControlAddCallback());

            // Wait for the onTopicAdded() callback.
            manualResetEvent.WaitOne();
```

This example uses a `ManualResetEvent` to wait until the topic is successfully added. This approach is used to simplify the example and is not recommended for production clients.

The `AddTopic()` method requires a callback, which you must create.

7. Implement an instance of `ITopicControlAddCallback` as an inner class of your publishing client. Declare a `ManualResetEvent` instance.

```
     private static readonly ManualResetEvent
manualResetEvent = new ManualResetEvent(false);

       private class TopicControlAddCallback :
ITopicControlAddCallback
       {
            public void OnTopicAdded(string topicPath)
            {
                manualResetEvent.Set();
            }

            public void OnTopicAddFailed(string topicPath,
TopicAddFailReason reason)
            {
            }

            public void OnDiscard()
            {
            }
       }
```

`ManualResetEvent` is used to simplify this example by making it more synchronous. However, the Diffusion .NET API is designed to be most powerful when used asynchronously.

8. Back in the `Main` method, loop once a second updating the foo/counter topic with an incrementing count.

Use the `updateControl.Updater.Update()` method to update a topic without locking that topic.

```
            // Update the topic
            for (int i = 0; i < 1000; ++i)
```

```
            {
                // Use the non-exclusive updater to update the
topic without locking it
                updateControl.Updater.Update("foo/counter",
i.ToString(), new TopicUpdaterUpdateCallback());

                Thread.Sleep(1000);
            }
```

The `Update()` method requires a callback, which you must create.

9. Implement an instance of `ITopicUpdaterUpdateCallback` as an inner class of your publishing client.

```
        private class TopicUpdaterUpdateCallback :
ITopicUpdaterUpdateCallback
        {
            public void OnError(ErrorReason errorReason)
            {
            }

            public void OnSuccess()
            {
            }
        }
```

For the purposes of this example, the callback is a no-op. In your clients, ensure that you define the behavior for these events.

10. Compile and run your client.

Results

The client publishes a value to the foo/counter topic every second. You can subscribe to the foo/counter topic by creating a client to subscribe to the topic. For more information, see Getting started with .NET on page 397.

Full example

The completed publishing client class contains the following code:

```
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Topics;

namespace Examples {
    class Program
    {
        private static readonly ManualResetEvent
 manualResetEvent = new ManualResetEvent(false);

        private class TopicControlAddCallback :
 ITopicControlAddCallback
        {
            public void OnTopicAdded(string topicPath)
            {
                manualResetEvent.Set();
            }
```

```
            public void OnTopicAddFailed(string topicPath,
TopicAddFailReason reason)
            {
            }

            public void OnDiscard()
            {
            }
        }

        private class TopicUpdaterUpdateCallback :
ITopicUpdaterUpdateCallback
        {
            public void OnError(ErrorReason errorReason)
            {
            }

            public void OnSuccess()
            {
            }
        }

        static void Main( string[] args ) {
            // Connect using a principal with 'modify_topic'
and 'update_topic'
            // permissions
            var session =
Diffusion.Sessions.Principal("principal").Password("password").Open("ws:/

            // Get the TopicControl and TopicUpdateControl
feature
            var topicControl =
session.GetTopicControlFeature();
            var updateControl =
session.GetTopicUpdateControlFeature();

            // Create a single value topic 'foo/counter'
            topicControl.AddTopic("foo/counter",
TopicType.SINGLE_VALUE, new TopicControlAddCallback());

            // Wait for the onTopicAdded() callback.
            manualResetEvent.WaitOne();

            // Update the topic
            for (int i = 0; i < 1000; ++i)
            {
                // Use the non-exclusive updater to update
the topic without locking
                // it
                updateControl.Updater.Update("foo/counter",
i.ToString(), new TopicUpdaterUpdateCallback());

                Thread.Sleep(1000);
            }
        }
    }
}
```

Using the .NET Unified API

Considerations and capabilities that are specific to the .NET Unified API.

Certificates

Diffusion .NET clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .NET Framework.

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the .NET Framework, you must set up a trust store for the client and add the appropriate certificates to that trust store.

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use the **Microsoft Management Console** to import the certificate into the **Trusted Root Certification Authorities** folder. For more information, see https://msdn.microsoft.com/en-us/library/aa738659(v=vs.110).aspx

.NET examples

Examples that use the .NET Unified API.

- Reconnection example on page 405
- AuthenticationControl example on page 408
- ClientControl example on page 409
- Messaging example on page 411
- MessagingControl example on page 413
- Pings example on page 416
- Security example on page 417
- SecurityControl example on page 418
- SystemAuthenticationControl example on page 422
- TopicControl example on page 425
- Topics example on page 433
- TopicUpdateControl example on page 438

Reconnection example

The following example shows using the .NET Unified API to define the client's reconnection behavior.

```
using System;
using System.Threading.Tasks;
using NLog.Time;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Session;
using
 PushTechnology.ClientInterface.Client.Session.Reconnection;

namespace UCIStack.Examples
{
    /// <summary>
    /// These examples show how to configure and enable the
 reconnection feature of the API.
    /// Every method represents a different selfcontained
 example.
    /// </summary>
```

```
   public class ClientReconnection
   {
       /// <summary>
       /// Sets the reconnection timeout that represents the
duration in which
       /// the client is trying to reconnect to the server.
       /// If we are not reconnected after the timeout, the
client will close the session.
       /// </summary>
       public void SetReconnectionTimeout()
       {
           // The timeout is set in milliseconds and should be
high enough to
           // account for actual reconnection time
           var sessionFactory =
Diffusion.Sessions.ReconnectionTimeout(60000);
       }

       /// <summary>
       /// Sets the reconnection strategy which represents the
client behaviour in case of
       /// a connection loss.
       /// </summary>
       public void SetReconnectionStrategy()
       {
           // The abort strategy will not attempt reconnection
after a connection loss.
           // However it will switch into the
RECOVERING_RECONNECT session state for a brief moment and
           // then will be set to CLOSED_BY_SERVER.
           var sessionFactoryWithAbort =
Diffusion.Sessions.ReconnectionStrategy(ReconnectionStrategy.Abort);

           // The delayed strategy will attempt reconnection
after a connection loss but will
           // wait for 5 seconds until it attempts to
reconnect. Every reconnection try within the
           // reconnection timeout will have a brief 5 second
waiting period.
           var sessionFactoryWithDelay =
Diffusion.Sessions.ReconnectionStrategy(ReconnectionStrategy.Delayed);
       }

       /// <summary>
       /// Disables the reconnection feature.
       /// </summary>
       public void DisableReconnection()
       {
           // This will disable the reconnection feature
completely and instead of
           // switching to the RECOVERING_RECONNECT session
state it will switch
           // straight to CLOSED_BY_SERVER.
           var sessionFactoryNoReconnection =
Diffusion.Sessions.NoReconnection();

           // This call has exactly the same effect as the
above statement.
           var sessionFactoryNoTimeout =
Diffusion.Sessions.ReconnectionTimeout(0);
       }

       /// <summary>
```

```
        /// This is a custom reconnection strategy that will try
to reconnect
        /// to the server up to 3 times and then abort.
        /// </summary>
        public class MyReconnectionStrategy :
IReconnectionStrategy
        {
            private int counter = 0;

            /// <summary>
            /// Here we put our actual reconnection logic. The
async keyword should always be added
            /// since it makes things easier for a void return
type.
            /// </summary>
            /// <param name="reconnectionAttempt">The
reconnection attempt wil be given by the session.</param>
            public async Task
PerformReconnection(IReconnectionAttempt reconnectionAttempt)
            {
                ++counter;
                if (counter <= 3)
                {
                    // We start the next reconnection attempt
                    reconnectionAttempt.Start();
                }
                else
                {
                    counter = 0;

                    // We abort any other reconnection attempt
and let the session switch to CLOSED_BY_SERVER.
                    reconnectionAttempt.Abort();
                }
            }
        }

        /// <summary>
        /// This applies the custom reconnection strategy.
        /// </summary>
        public void CustomReconnectionStrategy()
        {
            // We don't need to hold a reference to the
reconnection strategy
            var sessionFactoryWithCustomStrategy
= Diffusion.Sessions.ReconnectionStrategy(new
MyReconnectionStrategy());
        }

        /// <summary>
        /// Reconnection can be observed via session state
changes within the SessionStateChangeHandler.
        /// </summary>
        public void ObserveReconnection()
        {
            var sessionFactory =
Diffusion.Sessions.SessionStateChangedHandler((sender, args) =>
            {
                if
(args.NewState.Equals(SessionState.RECOVERING_RECONNECT))
                {
                    // This will be set on a connection loss and
indicates a reconnection attempt.
```

```
                    // Unless reconnection is disabled, at which
 point the session never gets switched
                    // to this state.
                    Console.WriteLine("We are in the process of
 reconnecting.");
                }
                else if
 (args.NewState.Equals(SessionState.CONNECTION_ATTEMPT_FAILED))
                {
                    // If a reconnection attempt fails because
 the server session timed out,
                    // we won't be able to reconnect anymore. At
 which point the session will
                    // switch to this state.
                    Console.WriteLine("We couldn't connect.");
                }
                else if
 (args.NewState.Equals(SessionState.CLOSED_BY_SERVER))
                {
                    // If the reconnection timeout is over, we
 will switch to this state.
                    // In case of disabled reconnection we will
 switch directly to this
                    // state on a connection loss.
                    Console.WriteLine("We lost connection.");
                }
                else if
 (args.NewState.Equals(SessionState.CONNECTED_ACTIVE))
                {
                    // This is the obvious state on our first
 connection. It is
                    // also the state to which we switch to
 after a successful reconnection attempt.
                    Console.WriteLine("We are connected.");
                }
            });
        }
    }
}
```

AuthenticationControl example

The following example shows using the .NET Unified API to register a control authentication handler with the Diffusion server. The example also includes a simple or empty authentication handler.

The name by which the control authentication handler is registered must be configured in the `Server.xml` configuration file of the Diffusion server for the control authentication handler to be called to handle authentication requests.

```
using System;
using System.Linq;
using System.Threading;
using PushTechnology.ClientInterface.Client.Details;
using PushTechnology.ClientInterface.Client.Factories;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a control client which registers an authentication
 handler with a server.
 /// </summary>
```

```
 public class ControlAuthenticationClient
 {
  /// <summary>
  /// Main entry point.
  /// </summary>
  /// <param name="args">The command line arguments, currently
 ignored.</param>
  public static void Main( string[] args )
  {
   var session = Diffusion.Sessions
    .Principal( "auth" )
    .Password( "auth_secret" )
    .Open( "ws://diffusion.example.com:80" );


 session.GetAuthenticationControlFeature().SetAuthenticationHandler(
    "control-client-auth-handler-example",

 Enum.GetValues( typeof( DetailType ) ).OfType<DetailType>().ToList(),
    new ExampleControlAuthenticationHandler() );

   while( true )
   {
    Thread.Sleep( 60000 );
   }
   // ReSharper disable once FunctionNeverReturns
  }
 }
}
```

ClientControl example

The following example shows using the .NET Unified API to set queue thresholds and register listeners for queue events.

```

using PushTechnology.ClientInterface.Client.Enums;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients;
using PushTechnology.ClientInterface.Client.Session;
using
 PushTechnology.ClientInterface.CommandServices.Commands.Control.Client;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates the use of a control client to apply both
 throttling and conflation to clients.  It throttles
 /// and conflates all clients that reach their queue thresholds
 and remove when they go down again.
 ///
 /// This uses the <see cref="IClientControl"/> feature.
 /// </summary>
 public class ControlClientConflateAndThrottle
 {
  #region Fields

  private readonly ISession session;

  #endregion Fields
```

```
 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 /// <param name="callback">Notifies callback from throttle
requests.</param>
 public ControlClientConflateAndThrottle( IClientCallback
callback )
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  // Create the client control feature with a handler that sets
queue thresholds on new connecting clients
  // and sets a listener for queue events.
  var clientControl = session.GetClientControlFeature();

  // To register a queue event handler, the client session must
have the REGISTER_HANDLER and VIEW_SESSION
  // permissions.
  clientControl.SetQueueEventHandler( new
MyThresholdHandler( clientControl, callback ) );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Close the session.
 /// </summary>
 public void Close()
 {
  session.Close();
 }

 #endregion Public Methods

 #region Private Classes

 private class MyThresholdHandler : QueueEventHandlerDefault
 {
  #region Fields

  private readonly IClientControl theClientControl;
  private readonly IClientCallback theClientCallback;

  #endregion Fields

  #region Constructor

  public MyThresholdHandler( IClientControl clientControl,
IClientCallback clientCallback )
  {
   theClientControl = clientControl;
   theClientCallback = clientCallback;
  }

  #endregion Constructor

  #region Overrides
```

```
    /// <summary>
    /// The configured upper queue threshold for a client's queue
 has been reached.
    /// </summary>
    /// <param name="client">The client session identifier.</
param>
    /// <param name="policy">The message queue policy.</param>
    public override void OnUpperThresholdCrossed( SessionId
 client, IMessageQueuePolicy policy )
    {
     // The SetThrottled method enables throttling and
 conflation.  This method requires the client session
     // to have the MODIFY_SESSION topic permission.
     theClientControl.SetThrottled( client,
 ThrottlerType.MESSAGE_INTERVAL, 10, theClientCallback );
    }

    /// <summary>
    /// The configured lower threshold for a client's queue has
 been reached.
    /// </summary>
    /// <param name="client">The client session identifier.</
param>
    /// <param name="policy">The message queue policy.</param>
    public override void OnLowerThresholdCrossed( SessionId
 client, IMessageQueuePolicy policy )
    {
     // The SetThrottled method enables throttling and
 conflation.  This method requires the client session
     // to have the MODIFY_SESSION topic permission.
     theClientControl.SetThrottled( client,
 ThrottlerType.MESSAGE_INTERVAL, 1000, theClientCallback );
    }

    #endregion Overrides
   }

  #endregion Private Classes
 }
}
```

Messaging example

The following example shows using the .NET Unified API to send a message to a topic.

```
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a simple example of a client that uses the
 'Messaging' feature to send messages on a topic path.
 ///
 /// To send messages on a topic path,
 the client session requires the <see
 cref="TopicPermission.SEND_TO_MESSAGE_HANDLER"/>
 /// permission.
```

```
/// </summary>
public class ClientSendingMessages
{
 #region Fields

 private readonly ISession session;
 private readonly IMessaging messaging;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructs a message sending application.
 /// </summary>
 public ClientSendingMessages()
 {
  session =
Diffusion.Sessions.Principal( "client" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  messaging = session.GetMessagingFeature();
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Sends a simple string message to a specified topic path.
 ///
 /// There will be no context with the message so callback will
be directed to the 'no context' callback.
 /// </summary>
 /// <param name="topicPath">The topic path.</param>
 /// <param name="message">The message to send.</param>
 /// <param name="callback">Notifies that the message was
sent.</param>
 public void Send( string topicPath, string message,
ISendCallback callback )
 {
  messaging.Send( topicPath,
Diffusion.Content.NewContent( message ), callback );
 }

 /// <summary>
 /// Sends a simple string message to a specified topic path
with context string.
 ///
 /// The callback will be directed to the contextual callback
with the string provided.
 /// </summary>
 /// <param name="topicPath"></param>
 /// <param name="message"></param>
 /// <param name="context"></param>
 /// <param name="callback"></param>
 public void Send( string topicPath, string message, string
context, ISendContextCallback<string> callback )
 {
  messaging.Send( topicPath,
Diffusion.Content.NewContent( message ), context, callback );
 }
```

```
  /// <summary>
  /// Sends a string message to a specified topic with headers.
  ///
  /// There will be no context with the message so callback will
 be directed to the 'no context' callback.
  /// </summary>
  /// <param name="topicPath">The topic path.</param>
  /// <param name="message">The message to send.</param>
  /// <param name="headers">The headers to send with the
 message.</param>
  /// <param name="callback">Notifies that the message was
 sent.</param>
  public void SendWithHeaders( string topicPath, string message,
 List<string> headers, ISendCallback callback )
  {
   messaging.Send(
    topicPath,
    Diffusion.Content.NewContent( message ),

 messaging.CreateSendOptionsBuilder().SetHeaders( headers ).Build(),
    callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

MessagingControl example

The following example shows using the .NET Unified API to receive messages sent to topics and to send messages to one or more clients

```
using System.Linq;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is an example of a control client using the <see
 cref="IMessagingControl"/> feature to receive messages
 /// from clients and also send messages to clients.
 ///
 /// It is a trivial example that simply responds to all
 messages on a particular branch of the topic tree by
 /// echoing them back to the client exactly as they are,
 complete with headers.
 /// </summary>
 public class ControlClientReceivingMessages
```

```
{
 #region Fields

 private readonly ISession session;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 /// <param name="callback">The callback to receive the result
of message sending.</param>
 public ControlClientReceivingMessages( ISendCallback
callback )
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  var messagingControl = session.GetMessagingControlFeature();

  // Register to receive all messages sent by clients on the
"foo" branch.
  // To do this, the client session must have the
REGISTER_HANDLER permission.
  messagingControl.AddMessageHandler( "foo", new
EchoHandler( messagingControl, callback ) );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Close the session.
 /// </summary>
 public void Close()
 {
  session.Close();
 }

 #endregion Public Methods

 #region Private Classes

 private class EchoHandler : MessageHandlerDefault
 {
  #region Fields

  private readonly IMessagingControl theMessagingControl;
  private readonly ISendCallback theSendCallback;

  #endregion Fields

  #region Constructor

  public EchoHandler( IMessagingControl messagingControl,
ISendCallback sendCallback )
  {
   theMessagingControl = messagingControl;
   theSendCallback = sendCallback;
```

```
   }

   #endregion Constructor

   #region Overrides

   /// <summary>
   /// Receives content sent from a session via a topic.
   /// </summary>
   /// <param name="sessionId">Identifies the client session
 that sent the content.</param>
   /// <param name="topicPath">The path of the topic that the
 content was sent on.</param>
   /// <param name="content">The content sent by the client.</
param>
   /// <param name="context">The context associated with the
 content.</param>
   public override void OnMessage(
    SessionId sessionId,
    string topicPath,
    IContent content,
    IReceiveContext context )
   {
    theMessagingControl.Send(
     sessionId,
     topicPath,
     content,
     theMessagingControl.CreateSendOptionsBuilder()
      .SetHeaders( context.HeadersList.ToList() )
      .Build(),
     theSendCallback );
   }

   #endregion Overrides
  }

  private class MessageHandlerDefault : TopicTreeHandlerDefault,
 IMessageHandler
  {
   /// <summary>
   /// Receives content sent from a session via a topic.
   /// </summary>
   /// <param name="sessionId">Identifies the client session
 that sent the content.</param>
   /// <param name="topicPath">The path of the topic that the
 content was sent on.</param>
   /// <param name="content">The content sent by the client.</
param>
   /// <param name="context">The context associated with the
 content.</param>
   public virtual void OnMessage( SessionId sessionId, string
 topicPath, IContent content, IReceiveContext context )
   {
   }
  }

  #endregion Private Classes
 }
}
```

Pings example

The following example shows using the .NET Unified API to ping Diffusion from a client.

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a simple client example that pings the server and
 prints out the round-trip time.
 ///
 /// This uses the <see cref="IPings"/> feature only.
 /// </summary>
 public class ClientUsingPings
 {
  #region Fields

  private readonly ISession session;
  private readonly IPings pings;

  #endregion Fields

  #region Constructor

  public ClientUsingPings()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   pings = session.GetPingFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Ping the server.
  /// </summary>
  /// <param name="context">The string to log with round-trip
 time.</param>
  /// <param name="callback">Used to return the ping reply.</
param>
  public void Ping( string context, IPingContextCallback<string>
 callback )
  {
   pings.PingServer( context, callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
```

```
}
```

Security example

The following example shows using the .NET Unified API to change the client principal and credentials for a client session.

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;

namespace UCIStack.Examples
{
 /// <summary>
 /// This demonstrates a client's use of credentials,
 specifically the ability to change the principal for an active
 /// session.
 ///
 /// This is not a realistic use case on its own, but it shown
 separately here for clarity.
 /// </summary>
 public class ClientUsingCredentials
 {
  #region Fields

  private readonly ISession session;
  private readonly ISecurity security;

  #endregion Fields

  #region Constructor

  public ClientUsingCredentials()
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   security = session.GetSecurityFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Request a change of principal for the session.
  /// </summary>
  /// <param name="principal">The new principal name.</param>
  /// <param name="password">The password.</param>
  /// <param name="callback">Notifies success or failure.</
param>
  public void ChangePrincipal( string principal, string
 password, IChangePrincipalCallback callback )
  {
   security.ChangePrincipal( principal,
 Diffusion.Credentials.Password( password ), callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
```

```
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

SecurityControl example

The following example shows using the .NET Unified API to edit the contents of the security store.

```
using System.Collections.Generic;
using System.Linq;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityCon
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
    /// <summary>
    /// An example of using a control client to alter the
 security configuration.
    ///
    /// This uses the <see cref="ISecurityControl"/> feature
 only.
    /// </summary>
    public class ControlClientChangingSecurity
    {
        #region Fields

        private readonly ISecurityControl securityControl;

        #endregion Fields

        #region Constructor

        public ControlClientChangingSecurity()
        {
            var session = Diffusion.Sessions
                // Authenticate with a user that has the
 VIEW_SECURITY and MODIFY_SECURITY permissions.
                .Principal( "admin" ).Password( "password" )
                // Use a secure channel because we're
 transferring sensitive information.
                .Open( "wss://diffusion.example.com:80" );

            securityControl =
 session.GetSecurityControlFeature();
        }

        #endregion Constructor

        #region Public Methods

        public void DoCapitalizeRoles( IUpdateStoreCallback
 callback )
        {
```

```
            securityControl.GetSecurity( new
 CapitalizeRoles( securityControl, callback ) );
        }

        #endregion Public Methods

        #region Private Classes

        private class CapitalizeRoles : IConfigurationCallback
        {
            #region Fields

            private readonly ISecurityControl
 theSecurityControl;
            private readonly IUpdateStoreCallback theCallback;

            #endregion Fields

            #region Constructor

            /// <summary>
            /// Constructor.
            /// </summary>
            /// <param name="securityControl">The security
 control object.</param>
            /// <param name="callback">The callback object.</
param>
            public CapitalizeRoles( ISecurityControl
 securityControl, IUpdateStoreCallback callback )
            {
                theSecurityControl = securityControl;
                theCallback = callback;
            }

            #endregion Constructor

            /// <summary>
            /// Notification of a contextual error related to
 this callback. This is
            /// analogous to an exception being raised.
 Situations in which
            /// <code>OnError</code> is called include the
 session being closed, a
            /// communication timeout, or a problem with the
 provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
                // This might fail if the session lacks the
 required permissions.
                theCallback.OnError( errorReason );
            }

            /// <summary>
            /// This is called to return the requested security
 configuration.
            /// </summary>
```

```
            /// <param name="configuration">The snapshot of
information from the security store.</param>
            public void OnReply( ISecurityConfiguration
configuration )
            {
                var builder =
theSecurityControl.ScriptBuilder();

                builder = builder.SetRolesForAnonymousSessions(

Capitalize( configuration.RolesForAnonymousSessions ) );

                builder = builder.SetRolesForNamedSessions(

Capitalize( configuration.RolesForNamedSessions ) );

                foreach( var role in configuration.Roles )
                {
                    var oldName = role.Name;
                    var newName = Capitalize( oldName );

                    // Only if new name is different
                    if( !oldName.Equals( newName ) )
                    {
                        // Global permissions
                        var globalPermissions =
role.GlobalPermissions;

                        if( globalPermissions.Count > 0 )
                        {
                            // Remove global permissions for old
role
                            builder =
builder.SetGlobalPermissions( oldName, new
List<GlobalPermission>() );

                            // Set global permissions for new
role
                            builder =
builder.SetGlobalPermissions( newName,
                                new
List<GlobalPermission>( role.GlobalPermissions ) );
                        }

                        var defaultTopicPermissions =
role.DefaultTopicPermissions;

                        if( defaultTopicPermissions.Count > 0 )
                        {
                            // Remove default topic permissions
for old role
                            builder =
builder.SetDefaultTopicPermissions( oldName, new
List<TopicPermission>() );

                            // Set default topic permissions for
new role
                            builder =
builder.SetDefaultTopicPermissions( newName,
                                new
List<TopicPermission>( role.DefaultTopicPermissions ) );
                        }
```

```
                        var topicPermissions =
role.TopicPermissions;

                        if( topicPermissions.Count > 0 )
                        {
                            foreach( var entry in
topicPermissions )
                            {
                                var topicPath = entry.Key;

                                // Remove old topic permissions
                                builder =
builder.RemoveTopicPermissions( oldName, topicPath );

                                // Set new topic permissions
                                builder =
builder.SetTopicPermissions( newName, topicPath, entry.Value );
                            }
                        }
                    }

                    var oldIncludedRoles = role.IncludedRoles;

                    if( oldIncludedRoles.Count > 0 )
                    {
                        // Remove old included roles
                        builder =
builder.SetRoleIncludes( oldName, new List<string>() );
                    }

                    // This is done even if role name did not
change as it is possible that roles included may have
                    var newIncludedRoles =
Capitalize( oldIncludedRoles );

                    builder = builder.SetRoleIncludes( newName,
newIncludedRoles );
                }
            }

            #region Private Methods

            private static List<string>
Capitalize( IEnumerable<string> roles )
            {
                return roles.Select( Capitalize ).ToList();
            }

            private static string Capitalize( string role )
            {
                return char.ToUpper( role[0] ) +
role.Substring( 1 );
            }

            #endregion Private Methods
        }

        #endregion Private Classes
    }
}
```

SystemAuthenticationControl example

The following example shows using the .NET Unified API to edit the contents of the system authentication store.

```
using System.Collections.Generic;
using System.Linq;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Clients;
using PushTechnology.ClientInterface.Client.Types;
using IUpdateStoreCallback =
 PushTechnology.ClientInterface.Client.Features.Control.Clients.SecurityCon

namespace UCIStack.Examples
{
    public class ControlClientChangingSystemAuthentication
    {
        #region Fields

        private readonly ISystemAuthenticationControl
 theSystemAuthenticationControl;

        #endregion Fields

        #region Constructor

        public ControlClientChangingSystemAuthentication()
        {
            var session =
                Diffusion.Sessions
                    // Authenticate with a user that has the
 VIEW_SECURITY and MODIFY_SECURITY permissions

 .Principal( "control" ).Password( "password" )
                    // Use a secure channel because we're
 transferring sensitive information.
                    .Open( "dpt://localhost:8080" );

            theSystemAuthenticationControl =
 session.GetSystemAuthenticationControlFeature();

            session.Start();
        }

        #endregion Constructor

        #region Public Methods

        /// <summary>
        /// For all system users, update the assigned roles to
 replace the 'SUPERUSER' role and with 'ADMINISTRATOR'.
        /// </summary>
        /// <param name="callback"></param>
        public void
 ChangeSuperUsersToAdministrators( IUpdateStoreCallback
 callback )
        {

 theSystemAuthenticationControl.GetSystemAuthentication(
```

```
                new
ChangeSuperusersToAdministrators( theSystemAuthenticationControl,
callback ) );
        }

        #endregion Public Methods

        #region Private Classes

        private class InternalUpdateStoreCallback :
IUpdateStoreCallback
        {
            /// <summary>
            /// The script was applied successfully.
            /// </summary>
            public void OnSuccess()
            {
            }

            /// <summary>
            /// The script was rejected.  No changes were made
to the system authentication store.
            /// </summary>
            /// <param name="errors">The details of why the
script was rejected.</param>
            public void OnRejected( ICollection<IErrorReport>
errors )
            {
            }

            /// <summary>
            /// Notification of a contextual error related to
this callback. This is
            /// analogous to an exception being raised.
Situations in which
            /// <code>OnError</code> is called include the
session being closed, a
            /// communication timeout, or a problem with the
provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
            }
        }

        private class ChangeSuperusersToAdministrators :
IConfigurationCallback
        {
            #region Fields

            private readonly ISystemAuthenticationControl
theSystemAuthenticationControl;
            private readonly IUpdateStoreCallback theCallback;

            #endregion Fields

            #region Constructor
```

```
            public ChangeSuperusersToAdministrators(
                ISystemAuthenticationControl
systemAuthenticationControl,
                IUpdateStoreCallback callback )
            {
                theSystemAuthenticationControl =
systemAuthenticationControl;
                theCallback = callback;
            }

            #endregion Constructor

            /// <summary>
            /// The configuration callback reply.
            /// </summary>
            /// <param
name="systemAuthenticationConfiguration">The system
authenticationConfiguration stored on the server.</param>
            public void
OnReply( ISystemAuthenticationConfiguration
systemAuthenticationConfiguration )
            {
                var builder =
theSystemAuthenticationControl.ScriptBuilder();

                // For all system users...
                foreach( var principal in
systemAuthenticationConfiguration.Principals )
                {
                    var assignedRoles = principal.AssignedRoles;

                    // ...that have the 'SUPERUSER' assigned
role...
                    if( !assignedRoles.Contains( "SUPERUSER" ) )
continue;

                    var newRoles = new
HashSet<string>( assignedRoles );

                    newRoles.Remove( "SUPERUSER" );
                    newRoles.Add( "ADMINISTRATOR" );

                    // ...and add a command to the script that
updates the user's assigned roles, replacing 'SUPERUSER' with
                    // 'ADMINISTRATOR'.
                    builder =
builder.AssignRoles( principal.Name, newRoles.ToList() );
                }

                var script = builder.Script();


theSystemAuthenticationControl.UpdateStore( script,
theCallback );
            }

            /// <summary>
            /// Notification of a contextual error related to
this callback. This is
            /// analogous to an exception being raised.
Situations in which
```

```
            /// <code>OnError</code> is called include the
 session being closed, a
            /// communication timeout, or a problem with the
 provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public void OnError( ErrorReason errorReason )
            {
                theCallback.OnError( errorReason );
            }
        }

        #endregion Private Classes

        #region Tests

        //[Test]
        public void
 ControlClientChangingSystemAuthenticationTest()
        {
            var client = new
 ControlClientChangingSystemAuthentication();

            client.ChangeSuperUsersToAdministrators( new
 InternalUpdateStoreCallback() );
        }

        #endregion Tests
    }
}
```

TopicControl example

The following example shows using the .NET Unified API to create topics.

```
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Content.Metadata;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client to add topics.
 ///
 /// This uses the <see cref="ITopicControl"/> feature only.
 ///
 /// To add or remove topics, the client session must have the
 <see cref="TopicPermission.MODIFY_TOPIC"/> permission
 /// for that branch of the topic tree.
 /// </summary>
```

```
public class ControlClientAddingTopics
{
 #region Fields

 private readonly ISession session;
 private readonly ITopicControl topicControl;

 #endregion Fields

 #region Constructor

 public ControlClientAddingTopics()
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  topicControl = session.GetTopicControlFeature();
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Adds a topic with the type derived from the value.
 ///
 /// This uses the simple convenience method for adding topics
where the topic type and metadata are derived
 /// from a supplied value which can be any object.  For
example, an integer would result in a single value topic
 /// of type 'integer'.
 /// </summary>
 /// <typeparam name="T">The value type.</typeparam>
 /// <param name="topicPath">The full topic path.</param>
 /// <param name="initialValue">An optional initial value for
the topic.</param>
 /// <param name="context">This will be passed back to the
callback when reporting success or failure of the
 /// topic add.</param>
 /// <param name="callback">To notify the result of the
operation.</param>
 /// <returns>The topic details used to add the topic.</
returns>
 public ITopicDetails AddTopicForValue<T>(
  string topicPath,
  T initialValue,
  string context,
  ITopicControlAddContextCallback<string> callback )
 {
  return topicControl.AddTopicFromValue( topicPath,
initialValue, context, callback );
 }

 /// <summary>
 /// Add a record topic from a list of initial values.
 ///
 /// This demonstrates the simplest mechanism for adding a
record topic by supplying values that both the
 /// metadata and the initial values are derived from.
 /// </summary>
 /// <param name="topicPath">The full topic path.</param>
```

```
  /// <param name="initialValues">The initial values for the
 topic fields which will also be used to derive the
  /// metadata definition of the topic.</param>
  /// <param name="context">This will be passed back to the
 callback when reporting success or failure of the
  /// topic add.</param>
  /// <param name="callback">To notify the result of the
 operation.</param>
  /// <returns></returns>
  public ITopicDetails AddRecordTopic(
   string topicPath,
   List<string> initialValues,
   string context,
   ITopicControlAddContextCallback<string> callback )
  {
   return topicControl.AddTopicFromValue(
    topicPath,

 Diffusion.Content.NewBuilder<IRecordContentBuilder>().PutFields( initialVa
    context,
    callback );
  }

  /// <summary>
  /// Adds a record topic with supplied metadata and optional
 initial content.
  ///
  /// This example shows details being created and would be fine
 when creating topics that are all different, but
  /// if creating many record topics with the same details, then
 it is far more efficient to pre-create the
  /// details.
  /// </summary>
  /// <param name="topicPath">The full topic path.</param>
  /// <param name="metadata">The pre-created record metadata.</
param>
  /// <param name="initialValue">The optional initial value for
 the topic which must have been created to match
  /// the supplied metadata.</param>
  /// <param name="context">The context passed back to the
 callback when the topic is created.</param>
  /// <param name="callback">To notify the result of the
 operation.</param>
  public void AddRecordTopic(
   string topicPath,
   IMContent metadata,
   IContent initialValue,
   string context,
   ITopicControlAddContextCallback<string> callback )
  {
   var details =
 topicControl.CreateDetailsBuilder<IRecordTopicDetailsBuilder>().Metadata(

   topicControl.AddTopic( topicPath, details, initialValue,
 context, callback );
  }

  /// <summary>
  /// Remove a single topic given its path.
  /// </summary>
  /// <param name="topicPath">The topic path.</param>
  /// <param name="callback">Notifies the result of the
 operation.</param>
```

```
  public void RemoveTopic( string topicPath,
 ITopicControlRemoveCallback callback )
  {
   topicControl.RemoveTopics( ">" + topicPath, callback );
  }

  /// <summary>
  /// Remove one or more topics using a topic selector
 expression.
  /// </summary>
  /// <param name="topicSelector">The selector expression.</
param>
  /// <param name="callback">Notifies the result of the
 operation.</param>
  public void RemoveTopics( string topicSelector,
 ITopicControlRemoveCallback callback )
  {
   topicControl.RemoveTopics( topicSelector, callback );
  }

  /// <summary>
  /// Request that the topic path and its descendants be removed
 when the session is closed (either explicitly
  /// using <see cref="ISession.Close"/>, or by the server).  If
 more than one session calls this method for the
  /// same topic path, the topics will be removed when the last
 session is closed.
  ///
  /// Different sessions may call this method for the same topic
 path, but not for topic paths above or below
  /// existing registrations on the same branch of the topic
 tree.
  /// </summary>
  /// <param name="topicPath">The part of the topic tree to
 remove when the last session is closed.</param>
  public void RemoveTopicsWithSession( string topicPath )
  {
   topicControl.RemoveTopicsWithSession( topicPath, new
 DefaultTopicTreeHandler() );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

The following example shows using the .NET Unified API to create paged topics.

```
using System;
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
```

```
using PushTechnology.ClientInterface.Client.Topics.Update;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client to create and update
 paged topics.
 ///
 /// This uses the <see cref="ITopicControl"/> feature to create
 a paged topic and the <see cref="ITopicUpdateControl"/>
 /// feature to send updates to it.
 ///
 /// This demonstrates some simple examples of paged topic
 updates but not all of the possible ways in which they
 /// can be done.
 ///
 /// To send updates to a topic, the client session requires the
 UPDATE_TOPIC permission for that branch of the
 /// topic tree.
 /// </summary>
 public class ControlClientUpdatingPagedTopics
 {
  #region Fields

  private const string OrderedTopic = "Paged/Ordered";
  private const string UnorderedTopic = "Paged/Unordered";

  private readonly ISession session;
  private readonly ITopicControl topicControl;
  private readonly IPagedRecordOrderedUpdateFactory
 orderedUpdateFactory;
  private readonly IPagedStringUnorderedUpdateFactory
 unorderedUpdateFactory;
  private static ITopicUpdater _pagedUpdater;

  #endregion Fields

  #region Constructor

  /// <summary>
  /// Constructor.
  /// </summary>
  public ControlClientUpdatingPagedTopics()
  {
   session =
 Diffusion.Sessions.Principal( "control" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topicControl = session.GetTopicControlFeature();
   var updateControl = session.GetTopicUpdateControlFeature();

   orderedUpdateFactory =
 updateControl.UpdateFactory<IPagedRecordOrderedUpdateFactory>();
   unorderedUpdateFactory =
 updateControl.UpdateFactory<IPagedStringUnorderedUpdateFactory>();

   var metadata = Diffusion.Metadata;

   // Create an unordered paged string topic
   topicControl.AddTopic(
    UnorderedTopic,
    topicControl.NewDetails( TopicType.PAGED_STRING ),
    new TopicControlAddCallbackDefault() );
```

```
  // Create an ordered paged record topic
  var recordMetadata = metadata.Record(
   "Record",
   metadata.String( "Name" ),
   metadata.String( "Address" ) );

  topicControl.AddTopic(
   OrderedTopic,
   topicControl
    .CreateDetailsBuilder<IPagedRecordTopicDetailsBuilder>()
    .Metadata( recordMetadata )
    .Order( new PagedRecordOrderKey( "Name" ) ).Build(),
   new TopicControlAddCallbackDefault() );

  // Register an updater for topics under the 'Paged' branch
  updateControl.RegisterUpdateSource( "Paged", new
UpdateSource() );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Add a new line from an ordered topic.
 /// </summary>
 /// <param name="name">The name field value.</param>
 /// <param name="address">The address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void AddOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
   orderedUpdateFactory.Add( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Update a line of an ordered topic.
 /// </summary>
 /// <param name="name">The name of the line to update.</param>
 /// <param name="address">The new address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void UpdateOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,

orderedUpdateFactory.Update( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Remove a line from an ordered topic.
 /// </summary>
 /// <param name="name">The name of the line to remove.</param>
```

```
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveOrdered( string name,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,

 orderedUpdateFactory.Remove( Diffusion.Content.NewRecord( name,
 "" ) ),
    callback );
  }

  /// <summary>
  /// Add a line or lines to the end of an unordered topic.
  /// </summary>
  /// <param name="values">The lines to add.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void AddUnordered( ICollection<string> values,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Add( values ),
    callback );
  }

  /// <summary>
  /// Insert a line or lines at a specified index within an
 unordered topic.
  /// </summary>
  /// <param name="index">The index at which to add the line.</
param>
  /// <param name="values">The lines to insert.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void InsertUnordered( int index, ICollection<string>
 values, ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Insert( index, values ),
    callback );
  }

  /// <summary>
  /// Remove a specific line from an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to remove.</
param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveUnordered( int index,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Remove( index ),
    callback );

  }
```

```
  /// <summary>
  /// Update a line within an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to update.</
param>
  /// <param name="value">The new line value.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void UpdateUnordered( int index, string value,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    OrderedTopic,
    unorderedUpdateFactory.Update( index, value ),
    callback );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   // Remove our topics and close the session when done.
   topicControl.RemoveTopics( ">Paged", new
 RemoveCallback( session ) );
  }

  #endregion Public Methods

  #region Private Methods

  private static void Update( string topic, IUpdate update,
 ITopicUpdaterUpdateCallback callback )
  {
   if( _pagedUpdater == null )
   {
    throw new InvalidOperationException( "The paged updater has
 not been initialised." );
   }

   _pagedUpdater.Update( topic, update, callback );
  }

  #endregion Private Methods

  #region Private Classes

  private class RemoveCallback :
 TopicControlRemoveCallbackDefault
  {
   #region Fields

   private readonly ISession theSession;

   #endregion Fields

   #region Constructor

   public RemoveCallback( ISession session )
   {
    theSession = session;
   }
```

```
    #endregion Constructor

    #region Overrides

    /// <summary>
    /// Notification that a call context was closed prematurely,
  typically due to a timeout or the session being
    /// closed.  No further calls will be made for the context.
    /// </summary>
    public override void OnDiscard()
    {
     theSession.Close();
    }

    /// <summary>
    /// Topic(s) have been removed.
    /// </summary>
    public override void OnTopicsRemoved()
    {
     theSession.Close();
    }

    #endregion Overrides
   }

   private class UpdateSource : TopicUpdateSourceDefault
   {
    #region Overrides

    /// <summary>
    /// State notification that this source is now active for the
  specified topic path, and is therefore in a valid
    /// state to send updates on topics at or below the
  registered topic path.
    /// </summary>
    /// <param name="topicPath">The registration path.</param>
    /// <param name="updater">An updater that may be used to
  update topics at or below the registered path.</param>
    public override void OnActive( string topicPath,
  ITopicUpdater updater )
    {
     _pagedUpdater = updater;
    }

    #endregion Overrides
   }

   #endregion Private Classes
  }
}
```

Topics example

The following example shows using the .NET Unified API to subscribe to topics and receive the topic content.

```
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Content;
using PushTechnology.ClientInterface.Client.Content.Metadata;
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
```

```
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
    /// <summary>
    /// In this simple and commonest case for a client, we just
 subscribe to a few topics and assign handlers for each
    /// to receive content.
    ///
    /// This makes use of the <see cref="ITopics"/> feature
 only.
    ///
    /// To subscribe to a topic, the client session must have
 the <see cref="TopicPermission.READ_TOPIC"/> permission
    /// for that branch of the topic tree.
    /// </summary>
    public class ClientSimpleSubscriber
    {
        #region Fields

        private readonly ISession session;

        #endregion Fields

        #region Constructor

        /// <summary>
        /// Constructor.
        /// </summary>
        public ClientSimpleSubscriber()
        {
            session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
                .Open( "ws://diffusion.example.com:80" );

            // Use the Topics feature to add a topic stream for
 Foo and all topics under Bar and request subscription
            // to those topics
            var topics = session.GetTopicsFeature();

            topics.AddTopicStream( ">Foo", new
 FooTopicStream() );
            topics.AddTopicStream( ">Bar/", new
 BarTopicStream() );


 topics.Subscribe( Diffusion.TopicSelectors.AnyOf( "Foo",
 "Bar//" ), new TopicsCompletionCallbackDefault() );
        }

        #endregion Constructor

        #region Public Methods

        /// <summary>
        /// Close the session.
        /// </summary>
        public void Close()
        {
            session.Close();
        }
```

```
        #endregion Public Methods

        #region Private Classes

        /// <summary>
        /// The topic stream for all messages on the 'Foo'
 topic.
        /// </summary>
        private class FooTopicStream : TopicStreamDefault
        {
            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public override void OnTopicUpdate( string
 topicPath, IContent content, IUpdateContext context )
            {
            }
        }

        /// <summary>
        /// The topic stream for all messages on 'Bar' topics.
        /// </summary>
        private class BarTopicStream : TopicStreamDefault
        {
            #region Fields

            #endregion Fields

            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public override void OnTopicUpdate( string
 topicPath, IContent content, IUpdateContext context )
            {
```

```
                var reader =
 Diffusion.Content.NewReader<IRecordContentReader>( content );

                foreach( var field in reader.NextRecord() )
                {
                    // Log the value of each field here.
                }
            }
        }

        private class TopicStreamDefault : ITopicStream
        {
            /// <summary>
            /// Notification of a contextual error related to
 this callback. This is
            /// analogous to an exception being raised.
 Situations in which
            /// <code>OnError</code> is called include the
 session being closed, a
            /// communication timeout, or a problem with the
 provided parameters. No
            /// further calls will be made to this callback.
            /// </summary>
            /// <param name="errorReason">errorReason a value
 representing the error; this can be one of
            /// constants defined in <see cref="ErrorReason" />,
 or a feature-specific
            /// reason.</param>
            public virtual void OnError( ErrorReason
 errorReason )
            {
            }

            /// <summary>
            /// Called to notify that a stream context was
 closed normally.
            ///
            /// No further calls will be made for the stream
 context.
            /// </summary>
            public void OnClose()
            {
            }

            /// <summary>
            /// This notifies when a topic is subscribed to.
            ///
            /// This provides only <see
 cref="TopicDetailsLevel.BASIC"/>details of the topic.
            /// </summary>
            /// <param name="topicPath">the full topic path</
param>
            /// <param name="details">the basic details</param>
            public virtual void OnSubscription( string
 topicPath, ITopicDetails details )
            {
            }

            /// <summary>
            /// This notifies when a topic is unsubscribed.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
```

```
            /// <param name="reason">the reason for
 unsubscription.</param>
            public virtual void OnUnsubscription( string
 topicPath, TopicUnsubscribeReason reason )
            {
            }

            /// <summary>
            /// Topic update received.
            ///
            /// This indicates an update to the state of a topic
 that is subscribed
            /// to.
            /// </summary>
            /// <param name="topicPath">the full topic path.</
param>
            /// <param name="content">the topic content. The
 context may contain more
            /// information about the nature of the content</
param>
            /// <param name="context">the update context which
 may indicate whether the
            /// content represents the total state or a change
 to the state</param>
            public virtual void OnTopicUpdate( string topicPath,
 IContent content, IUpdateContext context )
            {
            }
        }

        #endregion Private Classes
    }
}
```

The following example shows using the .NET Unified API to fetch the current state of a topic without subscribing to the topic.

```
using PushTechnology.ClientInterface.Client.Factories;
using PushTechnology.ClientInterface.Client.Features;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.DiffusionCore.Messaging.Topic;

namespace UCIStack.Examples
{
 /// <summary>
 /// This is a simple example of a client that fetches the state
 of topics but does not subscribe to them.
 ///
 /// This makes use of the <see cref="ITopics"/> feature only.
 /// </summary>
 public class ClientUsingFetch
 {
  #region Fields

  private readonly ISession session;
  private readonly ITopics topics;

  #endregion Fields

  #region Constructor

  public ClientUsingFetch()
```

```
  {
   session =
 Diffusion.Sessions.Principal( "client" ).Password( "password" )
    .Open( "ws://diffusion.example.com:80" );

   topics = session.GetTopicsFeature();
  }

  #endregion Constructor

  #region Public Methods

  /// <summary>
  /// Issues a fetch request for a topic or selection of topics.
  /// </summary>
  /// <param name="topicSelector">A <see cref="TopicSelector"/>
 expression.</param>
  /// <param name="fetchContext">The context string to be
 returned with the fetch response(s).</param>
  /// <param name="stream">The callback for fetch responses.</
param>
  public void Fetch( string topicSelector, string fetchContext,
 IFetchContextStream<string> stream )
  {
   topics.Fetch( topicSelector, fetchContext, stream );
  }

  /// <summary>
  /// Close the session.
  /// </summary>
  public void Close()
  {
   session.Close();
  }

  #endregion Public Methods
 }
}
```

TopicUpdateControl example

The following example shows using the .NET Unified API to update topics.

```
using System.Threading;
using PushTechnology.ClientInterface.Client.Callbacks;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Types;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client as an event feed to a
 topic.
 ///
 /// This uses the <see cref="ITopicControl"/> feature to create
 a topic and the <see cref="ITopicUpdateControl"/>
 /// feature to send updates to it.
 ///
```

```
/// To send updates to a topic, the client session requires the
<see cref="TopicPermission.UPDATE_TOPIC"/>
/// permission for that branch of the topic tree.
/// </summary>
public class ControlClientAsUpdateSource
{
 #region Fields

 private const string TopicName = "Feeder";

 private readonly ISession session;
 private readonly ITopicControl topicControl;
 private readonly ITopicUpdateControl updateControl;
 private readonly ITopicUpdaterUpdateCallback updateCallback;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 /// <param name="callback">The callback for updates.</param>
 public
ControlClientAsUpdateSource( ITopicUpdaterUpdateCallback
callback )
 {
  updateCallback = callback;

  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com;80" );

  topicControl = session.GetTopicControlFeature();
  updateControl = session.GetTopicUpdateControlFeature();
 }

 #endregion Constructor

 #region Public Methods

 public void Start( IPriceProvider provider )
 {
  // Set up topic details
  var builder =
topicControl.CreateDetailsBuilder<ISingleValueTopicDetailsBuilder>();
  var details =
builder.Metadata( Diffusion.Metadata.Decimal( "Price" ) ).Build();

  // Declare a custom update source implementation.  When the
source is set as active, start a periodic task
  // to poll the provider every second and update the topic.
When the source is closed, stop the scheduled
  // task.
  var source = new UpdateSource( provider, updateCallback );

  // Create the topic.  When the callback indicates that the
topic has been created, register the topic
  // source for the topic
  topicControl.AddTopicFromValue( TopicName, details, new
AddCallback( updateControl, source ) );
 }
```

```
 public void Close()
 {
  // Remove our topic and close the session when done.
  topicControl.RemoveTopics( ">" + TopicName, new
RemoveCallback( session ));
 }

 #endregion Public Methods

 #region Private Classes

 private class RemoveCallback :
TopicControlRemoveCallbackDefault
 {
  #region Fields

  private readonly ISession theSession;

  #endregion Fields

  #region Constructor

  public RemoveCallback( ISession session )
  {
   theSession = session;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
  /// Notification that a call context was closed prematurely,
typically due to a timeout or the session being
  /// closed.  No further calls will be made for the context.
  /// </summary>
  public override void OnDiscard()
  {
   theSession.Close();
  }

  /// <summary>
  /// Topic(s) have been removed.
  /// </summary>
  public override void OnTopicsRemoved()
  {
   theSession.Close();
  }

  #endregion Overrides
 }

 private class AddCallback : TopicControlAddCallbackDefault
 {
  #region Fields

  private readonly ITopicUpdateControl updateControl;
  private readonly UpdateSource updateSource;

  #endregion Fields

  #region Constructor
```

```
  public AddCallback( ITopicUpdateControl updater, UpdateSource
source )
  {
   updateControl = updater;
   updateSource = source;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
  /// Topic has been added.
  /// </summary>
  /// <param name="topicPath">The full path of the topic that
was added.</param>
  public override void OnTopicAdded( string topicPath )
  {
   updateControl.RegisterUpdateSource( topicPath,
updateSource );
  }

  #endregion Overrides
 }

 private class UpdateSource : TopicUpdateSourceDefault
 {
  #region Fields

  private readonly IPriceProvider thePriceProvider;
  private readonly ITopicUpdaterUpdateCallback
theUpdateCallback;
  private readonly CancellationTokenSource cancellationToken =
new CancellationTokenSource();

  #endregion Fields

  #region Constructor

  public UpdateSource( IPriceProvider provider,
ITopicUpdaterUpdateCallback callback )
  {
   thePriceProvider = provider;
   theUpdateCallback = callback;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
  /// State notification that this source is now active for the
specified topic path, and is therefore in a valid
  /// state to send updates on topics at or below the
registered topic path.
  /// </summary>
  /// <param name="topicPath">The registration path.</param>
  /// <param name="updater">An updater that may be used to
update topics at or below the registered path.</param>
  public override void OnActive( string topicPath,
ITopicUpdater updater )
  {
   PeriodicTaskFactory.Start( () =>
```

```
    {
     updater.Update(
      TopicName,
 Diffusion.Content.NewContent( thePriceProvider.Price ),
 theUpdateCallback );
    }, 1000, cancelToken:cancellationToken.Token );
   }

   /// <summary>
   /// Called if the handler is closed. The handler will be
 closed if the
   /// session is closed after the handler has been registered,
 or if the
   /// handler is unregistered using <see
 cref="IRegistration.Close">close</see>.
   ///
   /// No further calls will be made for the handler.
   /// </summary>
   /// <param name="topicPath">the branch of the topic tree for
 which the handler was registered</param>
   public override void OnClose( string topicPath )
   {
    cancellationToken.Cancel();
   }

   #endregion Overrides
  }

  #endregion Private Classes

  #region Interfaces

  public interface IPriceProvider
  {
   /// <summary>
   /// Get the current price as a decimal string.
   /// </summary>
   string Price { get; }
  }

  #endregion Interfaces
 }
}
```

The following example shows using the .NET Unified API to update paged topics.

```
using System;
using System.Collections.Generic;
using PushTechnology.ClientInterface.Client.Factories;
using
 PushTechnology.ClientInterface.Client.Features.Control.Topics;
using PushTechnology.ClientInterface.Client.Session;
using PushTechnology.ClientInterface.Client.Topics;
using PushTechnology.ClientInterface.Client.Topics.Update;

namespace UCIStack.Examples
{
 /// <summary>
 /// An example of using a control client to create and update
 paged topics.
 ///
```

```
/// This uses the <see cref="ITopicControl"/> feature to create
a paged topic and the <see cref="ITopicUpdateControl"/>
/// feature to send updates to it.
///
/// This demonstrates some simple examples of paged topic
updates but not all of the possible ways in which they
/// can be done.
///
/// To send updates to a topic, the client session requires the
UPDATE_TOPIC permission for that branch of the
/// topic tree.
/// </summary>
public class ControlClientUpdatingPagedTopics
{
 #region Fields

 private const string OrderedTopic = "Paged/Ordered";
 private const string UnorderedTopic = "Paged/Unordered";

 private readonly ISession session;
 private readonly ITopicControl topicControl;
 private readonly IPagedRecordOrderedUpdateFactory
orderedUpdateFactory;
 private readonly IPagedStringUnorderedUpdateFactory
unorderedUpdateFactory;
 private static ITopicUpdater _pagedUpdater;

 #endregion Fields

 #region Constructor

 /// <summary>
 /// Constructor.
 /// </summary>
 public ControlClientUpdatingPagedTopics()
 {
  session =
Diffusion.Sessions.Principal( "control" ).Password( "password" )
   .Open( "ws://diffusion.example.com:80" );

  topicControl = session.GetTopicControlFeature();
  var updateControl = session.GetTopicUpdateControlFeature();

  orderedUpdateFactory =
updateControl.UpdateFactory<IPagedRecordOrderedUpdateFactory>();
  unorderedUpdateFactory =
updateControl.UpdateFactory<IPagedStringUnorderedUpdateFactory>();

  var metadata = Diffusion.Metadata;

  // Create an unordered paged string topic
  topicControl.AddTopic(
   UnorderedTopic,
   topicControl.NewDetails( TopicType.PAGED_STRING ),
   new TopicControlAddCallbackDefault() );

  // Create an ordered paged record topic
  var recordMetadata = metadata.Record(
   "Record",
   metadata.String( "Name" ),
   metadata.String( "Address" ) );

  topicControl.AddTopic(
```

```
    OrderedTopic,
    topicControl
     .CreateDetailsBuilder<IPagedRecordTopicDetailsBuilder>()
     .Metadata( recordMetadata )
     .Order( new PagedRecordOrderKey( "Name" ) ).Build(),
    new TopicControlAddCallbackDefault() );

   // Register an updater for topics under the 'Paged' branch
   updateControl.RegisterUpdateSource( "Paged", new
UpdateSource() );
 }

 #endregion Constructor

 #region Public Methods

 /// <summary>
 /// Add a new line from an ordered topic.
 /// </summary>
 /// <param name="name">The name field value.</param>
 /// <param name="address">The address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void AddOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
   orderedUpdateFactory.Add( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Update a line of an ordered topic.
 /// </summary>
 /// <param name="name">The name of the line to update.</param>
 /// <param name="address">The new address field value.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void UpdateOrdered( string name, string address,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,

orderedUpdateFactory.Update( Diffusion.Content.NewRecord( name,
address ) ),
   callback );
 }

 /// <summary>
 /// Remove a line from an ordered topic.
 /// </summary>
 /// <param name="name">The name of the line to remove.</param>
 /// <param name="callback">The callback to notify the
result.</param>
 public void RemoveOrdered( string name,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
```

```
 orderedUpdateFactory.Remove( Diffusion.Content.NewRecord( name,
 "" ) ),
    callback );
  }

  /// <summary>
  /// Add a line or lines to the end of an unordered topic.
  /// </summary>
  /// <param name="values">The lines to add.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void AddUnordered( ICollection<string> values,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Add( values ),
    callback );
  }

  /// <summary>
  /// Insert a line or lines at a specified index within an
 unordered topic.
  /// </summary>
  /// <param name="index">The index at which to add the line.</
param>
  /// <param name="values">The lines to insert.</param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void InsertUnordered( int index, ICollection<string>
 values, ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Insert( index, values ),
    callback );
  }

  /// <summary>
  /// Remove a specific line from an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to remove.</
param>
  /// <param name="callback">The callback to notify the
 result.</param>
  public void RemoveUnordered( int index,
 ITopicUpdaterUpdateCallback callback )
  {
   Update(
    UnorderedTopic,
    unorderedUpdateFactory.Remove( index ),
    callback );

  }

  /// <summary>
  /// Update a line within an unordered topic.
  /// </summary>
  /// <param name="index">The index of the line to update.</
param>
  /// <param name="value">The new line value.</param>
```

```
 /// <param name="callback">The callback to notify the
result.</param>
 public void UpdateUnordered( int index, string value,
ITopicUpdaterUpdateCallback callback )
 {
  Update(
   OrderedTopic,
   unorderedUpdateFactory.Update( index, value ),
   callback );
 }

 /// <summary>
 /// Close the session.
 /// </summary>
 public void Close()
 {
  // Remove our topics and close the session when done.
  topicControl.RemoveTopics( ">Paged", new
RemoveCallback( session ) );
 }

 #endregion Public Methods

 #region Private Methods

 private static void Update( string topic, IUpdate update,
ITopicUpdaterUpdateCallback callback )
 {
  if( _pagedUpdater == null )
  {
   throw new InvalidOperationException( "The paged updater has
not been initialised." );
  }

  _pagedUpdater.Update( topic, update, callback );
 }

 #endregion Private Methods

 #region Private Classes

 private class RemoveCallback :
TopicControlRemoveCallbackDefault
 {
  #region Fields

  private readonly ISession theSession;

  #endregion Fields

  #region Constructor

  public RemoveCallback( ISession session )
  {
   theSession = session;
  }

  #endregion Constructor

  #region Overrides

  /// <summary>
```

```
    /// Notification that a call context was closed prematurely,
  typically due to a timeout or the session being
    /// closed.  No further calls will be made for the context.
    /// </summary>
    public override void OnDiscard()
    {
     theSession.Close();
    }

    /// <summary>
    /// Topic(s) have been removed.
    /// </summary>
    public override void OnTopicsRemoved()
    {
     theSession.Close();
    }

    #endregion Overrides
   }

   private class UpdateSource : TopicUpdateSourceDefault
   {
    #region Overrides

    /// <summary>
    /// State notification that this source is now active for the
  specified topic path, and is therefore in a valid
    /// state to send updates on topics at or below the
  registered topic path.
    /// </summary>
    /// <param name="topicPath">The registration path.</param>
    /// <param name="updater">An updater that may be used to
  update topics at or below the registered path.</param>
    public override void OnActive( string topicPath,
  ITopicUpdater updater )
    {
     _pagedUpdater = updater;
    }

    #endregion Overrides
   }

   #endregion Private Classes
  }
}
```

JavaScript Unified API

The JavaScript Unified API is provided in the file `diffusion.js`.

The JavaScript library is provided with the Diffusion server installation in the `clients/js` folder.

Using the JavaScript Unified API

The Diffusion JavaScript API can be used directly in a web page or with Node.js.

To use the JavaScript API in a web page, include it with the `<script>` tag.

```
<script src="diffusion.js"></script>
```

To use the JavaScript API with Node.js, use `require`.

```
var diffusion = require('diffusion');
```

Support

Table 55: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
JavaScript Unified API	es5	• WS, WSS

Features are sets of capabilities provided by the API.

Table 56: Capabilities provided by the API

Feature set	Supported capabilities
Messaging	Send messages through a topic path to the message handler of that topic path Receive messages sent through a topic path
Messaging Control	Send messages to a specific client or set of clients Receive messages sent to the message handler of a topic path
Security	Change the security principal and credentials associated with the client session
Security Control	Define the permissions assigned to roles Define the roles assigned to anonymous sessions and named sessions.
System Authentication Control	Define users that are authenticated by the system authentication handler.
Topic Control	Create topics Delete topics
Topics	Subscribe to topics (excluding paged topics) Subscribe to a view of multiple topics Receive streaming updates on subscribed topics Unsubscribe from topics

Feature set	Supported capabilities
Topic Update Control	Update topics non-exclusively

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 57: Capabilities provided by the library

Library capability	Supported
Connect through load balancer	✓
Reconnect	✓

Getting started with JavaScript

Create a browser client within minutes that connects to the Diffusion server. This example creates a web page that automatically updates and displays the value of a topic.

Before you begin

To complete this example, you need a Diffusion server and a web server where you can host your client application.

You also require either a named user that has a role with the read_topic permission or that anonymous client connections are assigned a role with the read_topic permission. For example, the "CLIENT" role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

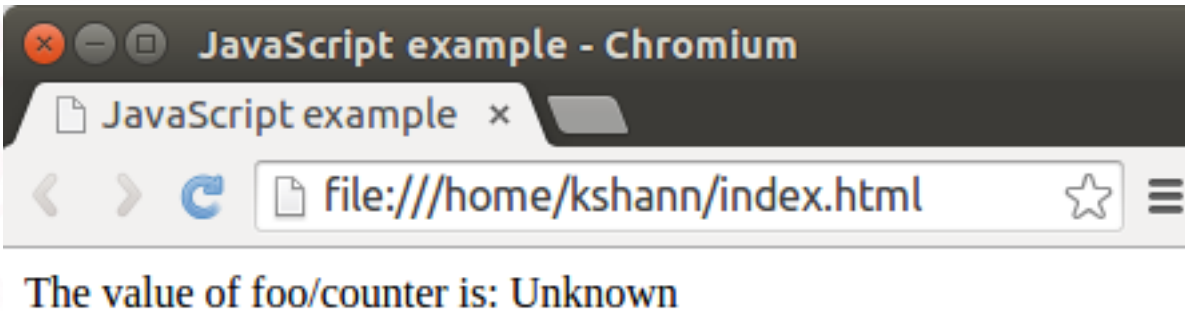
Procedure

1. Create a template HTML page which displays the information.

   For example, create the following `index.html` in your project's HTML directory.

```
<html>
  <head>
    <title>JavaScript example</title>
  </head>
  <body>
    <span>The value of foo/counter is: </span>
    <span id="update">Unknown</span>
  </body>
</html>
```

   If you open the page in a web browser it looks like the following screenshot:



2. Include the Diffusion JavaScript library in the <head> section of your `index.html` file.

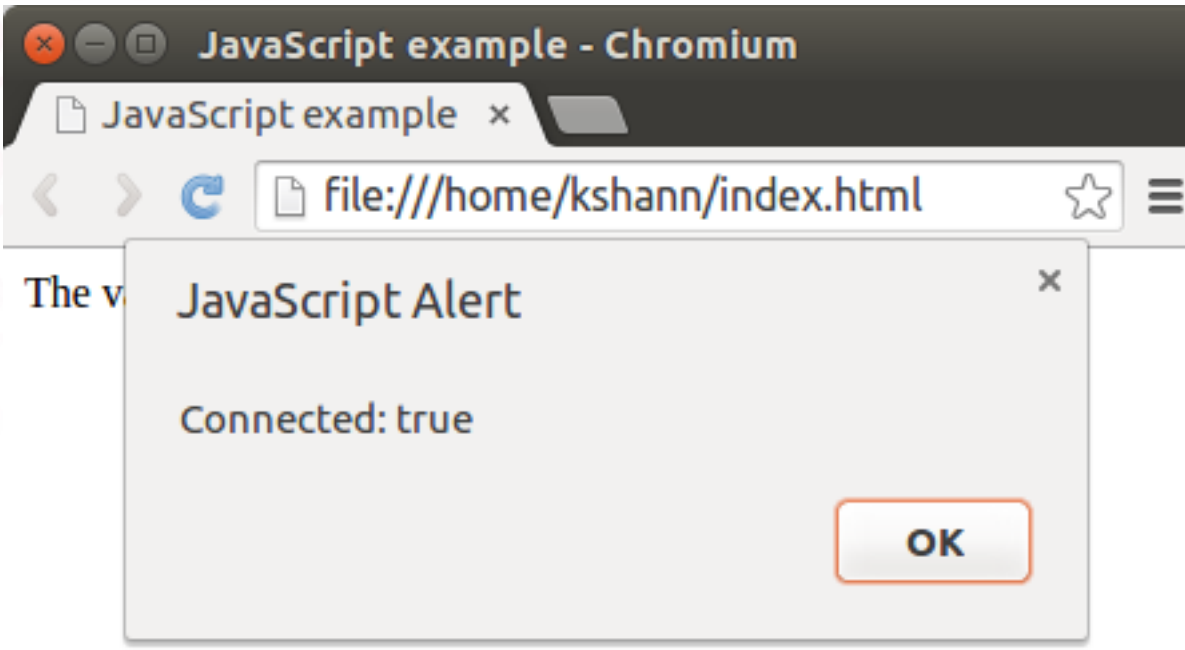
```
<head>
  <title>JavaScript example</title>
  <script type="text/javascript" src="http://
download.pushtechnology.com/clients/js/diffusion.js"></script>
</head>
```

3. Create a connection from the page to the Reappt service. Add a `script` element to the `body` element.

```
   <body>
    <span>The value of foo/counter is: </span>
    <span id="update">Unknown</span>
    <script type="text/javascript">
      diffusion.connect({
          // Edit this line to include the URL of your Reappt
 service
          host : 'service-name.reappt.io',
          // To connect anonymously you can leave out the
 following parameters
          principal : 'user',
          credentials : 'password'
      }).then(function(session) {
              alert('Connected: ' + session.isConnected());
          }
      );
    </script>
  </body>
```

Where *service-name* is the name of your Reappt service, *user* is the name of a user with the permissions required to subscribe to a topic, and *password* is the user's password.

If you open the page in a web browser it looks like the following screenshot:



4. Subscribe to a topic and receive data from it.

   Add the following function before the `diffusion.connect()` call:

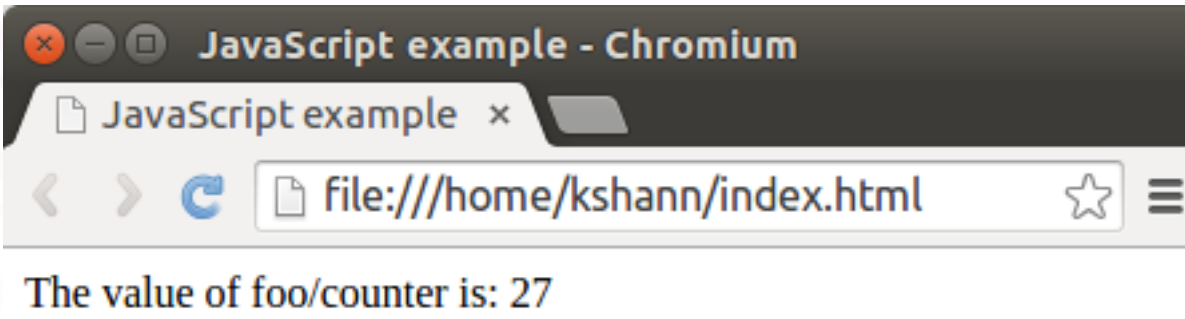
```
function subscribeToTopic(session) {
    session.subscribe('foo/counter')
        .on('update', function(data) {
            document.getElementById('update').innerHTML =
 data;
        }
    );
}
```

   The `subscribe()` method of the `session` object takes the name of the topic to subscribe to and emits an update event. The attached function takes the data from the topic and updates the `update` element of the web page with the topic data.

5. Change the function that is called on connection to the `subscribeToTopic` function you just created.

```
.then(subscribeToTopic);
```

   If you open the page in a web browser it looks like the following screenshot:



Results

The web page is updated every time the value of the foo/counter topic is updated. You can update the value of the foo/counter topic by creating a publishing client to update the topic. For more information, see Start publishing with JavaScript on page 453.

Example

The completed `index.html` file contains the following code:

```
<html>
  <head>
    <title>JavaScript example</title>

    <script type="text/javascript" src="http://
download.pushtecnology.com/clients/js/diffusion.js"></script>
  </head>
  <body>
    <div>
      <span>The value of foo/counter is: </span>
      <span id="update">Unknown</span>
    </div>

    <script type="text/javascript">
      function subscribeToTopic(session) {
          session.subscribe('foo/counter')
              .on('update', function(data) {
                  document.getElementById('update').innerHTML
 = data;
              }
          );
      }
      diffusion.connect({
          // Edit this line to include the URL of your Reappt
 service
          host : 'service-name.reappt.io',
          // To connect anonymously you can leave out the
 following parameters
```

```
            principal : 'user',
            credentials : 'password'
        }).then(subscribeToTopic);
    </script>
  </body>
</html>
```

Start publishing with JavaScript

Create a Node.js client that publishes data through topics on the Diffusion server.

Before you begin

To complete this example, you need a Diffusion server and a development system with Node.js and npm installed on it.

You also require either a named user that has a role with the modify_topic and update_topic permissions. For example, the "ADMINISTRATOR" role. For more information about roles and permissions, see Role-based authorization on page 134.

About this task

This example steps through the lines of code required to subscribe to a topic. The full code example is provided after the steps.

Procedure

1. Install the Diffusion JavaScript library on your development system.

   ```
   npm install diffusion
   ```

2. Create the JavaScript file that will be your publisher.

   For example, `publisher.js`

   a) Require the Diffusion library.

   ```
   var diffusion = require('diffusion');
   ```

   b) Connect to the Diffusion server.

   ```
   diffusion.connect({
       host : 'service-name.reappt.io',
       principal : 'control-user',
       credentials : 'password'
   }).then(function(session) {
       console.log('Connected!');
   });
   ```

   Where *service-name* is the name of your Reappt service, *control-user* is the name of a user with the permissions required to create and update topics, and *password* is the user's password.

   c) Create a topic called foo/counter and set its initial value to 0.

   ```
   session.topics.add('foo/counter', 0);
   ```

   d) Every second update the value of the topic with the value of the counter.

   ```
       setInterval(function() {
           session.topics.update('foo/counter', ++count);
       }, 1000);
   ```

3. Use Node.js to run your publishing client from the command line.

```
node publisher.js
```

Results

The publisher updates the value of the foo/counter topic every second. You can watch the topic value being updated by subscribing to the topic.

- You can use the example subscribing client from Getting started with JavaScript on page 449 to subscribe to foo/counter and output the value on a web page.

Example

The completed `publisher.js` file contains the following code:

```
var diffusion = require('diffusion');

diffusion.connect({
    host : 'service-name.reappt.io',
    principal : 'control-user',
    credentials : 'password'
}).then(function(session) {
    console.log('Connected!');

    var count = 0;

    // Create a topic with a default value of 0.
    session.topics.add('foo/counter', count);

    // Start updating the topic every second
    setInterval(function() {
        session.topics.update('foo/counter', ++count);
    }, 1000);
});
```

What to do next

Now that you have the outline of a publisher, you can use it to publish your own data instead of a counter.

Using the JavaScript Unified API

Considerations and capabilities that are specific to the JavaScript Unified API.

Using the JavaScript Unified API

The Diffusion JavaScript API can be used directly in a web page or with Node.js.

To use the JavaScript API in a web page, include it with the `<script>` tag.

```
<script src="diffusion.js"></script>
```

To use the JavaScript API with Node.js, use `require`.

```
var diffusion = require('diffusion');
```

Views

The JavaScript Unified API provides a view capability.

Use views to subscribe to multiple topics by using a topic selector and receive all the data from all topics in the selector set as a single structure when any of the topics are updated. If the topic selector matches a topic which is subsequently added or removed, the view is updated.

The following example shows views being used to present data from multiple topics as a single structure:

```
diffusion.connect({
    host        : 'localhost',
    port        : 8080,
    secure      : false,
    principal   : 'control',
    credentials : 'password'
}).then(function(session) {

    // Assuming a topic tree:
    //
    // scores
    //   |-- football
    //   |     |-- semi1
    //   |     |-- semi2
    //   |     |-- final
    //   |
    //   |-- tennis
    //         |-- semi1
    //         |-- semi2
    //         |-- final

    // Use a regular expression to create a view of the topics
 tracking the
    // scores during the finals for each sport.
    var view = session.view('?scores/.*/final');

    // Alternatively, we can use a topic set. Note that the
 topics do not need
    // to be under a common root, they may be anywhere within
 the topic tree.
    var view2 = session.view('#>scores/football/final////
>scores/tennis/final');

    // If any of the topics in the view change, display which
 topic changed
    // and its new value.
    view.on({
        update : function(value) {
            // Get and print the entire view structure.
            console.log('Update: ', JSON.stringify(value,
 undefined, 4));

            // Get individual topics. Returns a Buffer, which is
 automatically
            // converted to a String during concatenation,
 below.
            //
            // Note that the structure may not exist if the
 value has not been
            // updated.
```

```
            console.log('Football score: ' +
 value.scores.football.final);
            console.log('Tennis score  : ' +
 value.scores.tennis.final);

            // or ...
            // console.log('Football score: ' + value['scores']
['football']['final']);
        }
    });

    // The structure can also be accessed outside the update
 event.
    console.log('Football score: ' +
 view.get().scores.football.final);
});
```

Promises

The Diffusion JavaScript Unified API uses the Promises/A+ specification.

JavaScript examples

Examples that use the JavaScript Unified API.

Live code demos

You can see our live JavaScript demos here:

- Publisher demo
- Subscriber demo

Using the JavaScript Unified API

The Diffusion JavaScript API can be used directly in a web page or with Node.js.

To use the JavaScript API in a web page, include it with the `<script>` tag.

```
<script src="diffusion.js"></script>
```

To use the JavaScript API with Node.js, use `require`.

```
var diffusion = require('diffusion');
```

Node.js does not accept self-signed certificates by default. If you use a self-signed certificate, set the environment variable `NODE_TLS_REJECT_UNAUTHORIZED=0` before using any of these examples.

- Reconnection example on page 457
- Messaging and MessagingControl example on page 458
- Security, SecurityControl, and SystemAuthenticationControl example on page 459
- TopicControl examples on page 461
- Topics examples on page 463
- TopicUpdateControl example on page 466

Reconnection example

The following example shows using the JavaScript Unified API to define the client's reconnection behavior.

```
var diffusion = require('diffusion');

// When establishing a session, it is possible to specify
 whether reconnection
// should be attempted in the event of an unexpected
 disconnection. This allows
// the session to recover its previous state.

// Set the maximum amount of time we'll try and reconnect for to
 10 minutes
var maximumTimeoutDuration = 1000 * 60 * 10;

// Set the maximum interval between reconnect attempts to 60
 seconds
var maximumAttemptInterval = 1000 * 60;

// Create a reconnection strategy that applies an exponential
 back-off
var reconnectionStrategy = (function() {
    var attempts = 0;

    return function(start, abort) {
        var wait = Math.min(Math.pow(2, attempts++) * 100,
 maximumAttemptInterval);

        // Wait and then try to start the reconnection attempt
        setTimeout(start, wait);
    };
})();

// Connect to the server.
diffusion.connect({
    host : 'diffusion.example.com',
    port : 443,
    secure : true,
    principal : 'control',
    credentials : 'password',
    reconnect : {
        timeout : maximumTimeoutDuration,
        strategy : reconnectionStrategy
    }
}).then(function(session) {

    session.on('disconnect', function() {
        // This will be called when we lose connection. Because
 we've specified the
        // reconnection strategy, it will be called
 automatically when this event
        // is dispatched
    });

    session.on('reconnect', function() {
        // If the session is able to reconnect within the
 reconnect timeout, this
        // event will be dispatched to notify that normal
 operations may resume
    });
```

```
    session.on('close', function() {
        // If the session is closed normally, or the session is
 unable to reconnect,
        // this event will be dispatched to notify that the
 session is no longer
        // operational.
    });
});
```

Messaging and MessagingControl example

The following example shows using the JavaScript Unified API to send messages to sessions or topic paths and to register a message handler.

```
var diffusion = require('diffusion');

// Connect to the server. Change these options to suit your own
 environment.
// Node.js will not accept self-signed certificates by default.
 If you have
// one of these, set the environment variable
 NODE_TLS_REJECT_UNAUTHORIZED=0
// before running this example.
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. Messages can be sent & received between sessions.

    // Create a stream of received messages for a specific path
    session.messages.listen('foo').on('message', function(msg) {
        console.log('Received message: ' + msg.content);
    });

    // Send a message to another session. It is the
 application's responsibility to find the SessionID of the
 intended
    // recipient.
    session.messages.send('foo', 'Hello world', 'another-
session-id');

    // 2. Messages can also be sent without a recipient, in
 which case they will be dispatched to any Message Handlers
    // that have been registered for the same path. If multiple
 handlers are registered to the same path, any given
    // message will only be dispatched to one handler.

    // Register the handler to receive messages at or below the
 given path.
    session.messages.addHandler('foo', {
        onActive : function() {
            console.log('Handler registered');
        },
        onClose : function() {
            console.log('Handler closed');
        },
        onMessage : function(msg) {
```

```
            console.log('Received message:' + msg.content + '
 from Session: ' + msg.session);
            if (msg.properties) {
                console.log('with properties:', msg.properties);
            }
        }
    }).then(function() {
        console.log('Registered handler');
    }, function(e) {
        console.log('Failed to register handler: ', e);
    });

    // Send a message at a lower path, without an explicit
 recipient - this will be received by the Handler.
    session.messages.send('foo/bar', 'Another message');
});
```

Security, SecurityControl, and SystemAuthenticationControl example

The following example shows using the JavaScript Unified API to change the session principal, update the security information used by the Diffusion server, and update the system authentication information used by the system authentication handler.

```
// Session security allows you to change the principal that a
 session is authenticated as. It also  allows users to
// query and update server-side security and authentication
 stores, which control users, roles and permissions.
// This enables you to manage the capabilities that any logged
 in user will have access to.

// Connect to Diffusion with control client credentials
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {

    // 1. A session change their principal by re-authenticating
    session.security.changePrincipal('admin',
 'password').then(function() {
        console.log('Authenticated as admin');
    });

    // 2. The security configuration provides details about
 roles and their assigned permissions

 session.security.getSecurityConfiguration().then(function(config)
 {
        console.log('Roles for anonymous sessions: ',
 config.anonymous);
        console.log('Roles for named sessions: ', config.named);
        console.log('Available roles: ', config.roles);
    }, function(error) {
        console.log('Unable to fetch security configuration',
 error);
    });

    // 3. Changes to the security configuration are done with a
 SecurityScriptBuilder
```

```
    var securityScriptBuilder =
session.security.securityScriptBuilder();

    // Set the permissions for a particular role - global and
topic-scoped
    // Each method on a script builder returns a new builder
    var setPermissionScript =
securityScriptBuilder.setGlobalPermissions('SUPERUSER',
['REGISTER_HANDLER'])

.setTopicPermissions('SUPERUSER', '/foo', ['UPDATE_TOPIC'])
                                                    .build();

    // Update the server-side store with the generated script

session.security.updateSecurityStore(setPermissionScript).then(function()
{
        console.log('Security configuration updated
successfully');
    }, function(error) {
        console.log('Failed to update security configuration: ',
error);
    });

    // 4. The system authentication configuration lists all
users & roles

session.security.getSystemAuthenticationConfiguration().then(function(conf
{
        console.log('System principals: ', config.principals);
        console.log('Anonymous sessions: ', config.anonymous);
    }, function(error) {
        console.log('Unable to fetch system authentication
configuration', error);
    });

    // 5. Changes to the system authentication config are done
with a SystemAuthenticationScriptBuilder
    var authenticationScriptBuilder =
session.security.authenticationScriptBuilder();

    // Add a new user and set password & roles.
    var addUserScript =
authenticationScriptBuilder.addPrincipal('Superman',
'correcthorsebatterystapler')

.assignRoles('Superman', ['SUPERUSER'])
                                                    .build();

    // Update the system authentication store
    session.security.updateStore(addUserScript).then(function()
{
        console.log('Updated system authentication config');
    }, function(error) {
        console.log('Failed to update system authentication: ',
error);
    });
});
```

TopicControl examples

The following example shows using the JavaScript Unified API to create topics.

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. Topics can be created with a specified topic path and
 value. If the path contains multiple levels, any
    // intermediary topics that do not exist will be created
 automatically with a stateless topic type.

    // Create a topic with string values, and an initial value
 of "xyz".
    session.topics.add('topic/string', 'xyz');

    // Create a topic with integer values, and an initial value
 of 123.
    session.topics.add('topic/integer', 123);

    // Create a topic with decimal values, with an implicit
 scale of 2 and an initial value of 1.23.
    session.topics.add('topic/decimal', 1.23);

    // 2. Adding a topic returns a result, which allows us to
 handle when the operation has either
    // completed successfully or encountered an error.
    session.topics.add('topic/result',
 'abc').then(function(result) {
        console.log('Added topic: ' + result.topic);
    }, function(reason) {
        console.log('Failed to add topic: ', reason);
    });

    // Adding a topic that already exists will succeed, so long
 as it has the same value type
    session.topics.add('topic/result',
 'xyz').then(function(result) {
        // result.added will be false, as the topic already
 existed
        console.log('Added topic: ' + result.topic,
 result.added);
    });

    // Because the result returned from adding a topic is a
 promise, we can easily chain
    // multiple topic adds together
    session.topics.add('chain/foo',
 1).then(session.topics.add('chain/bar', 2))

 .then(session.topics.add('chain/baz', 3))

 .then(session.topics.add('chain/bob', 4))
                                      .then(function() {
                                         console.log('Added all
 topics');
                                      }, function(reason) {
                                         console.log('Failed to
 add topic', reason);
```

```
                                        });

   // 3. Metadata can be used to create topics that will
contain values of a specified format.

   // RecordContent formats data in a series of records and
fields, similar to tabular data.
   // Each record & field is named, allowing direct lookup of
values. Each field value has a
   // particular type (string, integer, decimal)
   var metadata = new diffusion.metadata.RecordContent();

   // Records are like rows in a table. They can have fields
assigned, with default values.
   // You can add fields all at once like this, or individually
(see below).
   var game = metadata.addRecord('game', 1, {
       'title' : metadata.string(),
       'round' : metadata.integer(0),
       'count' : metadata.integer(0)
   });

   // Records and fields can be set as occurring a certain
number of times.
   var player = metadata.addRecord('player', metadata.occurs(0,
8));

   // Add fields to a record individually.
   player.addField('name', 'Anonymous');
   player.addField('score', 0);

   // Adding the topic works just like normal.
   session.topics.add('games/some-game', metadata);

   // And the metadata can be re-used for multiple topics.
   session.topics.add('games/some-other-game', metadata);

   // 4. Using metadata, it is possible to create a topic with
both a metadata format, and the initial value

   // Topic values can be produced from metadata via the
builder interface
   var builder = metadata.builder();

   // Values must be set before a value can be created
   builder.add('game', { title : 'Planet Express!', count :
3 });

   builder.add('player', { name : 'Fry', score : 0 });
   builder.add('player', { name : 'Amy', score : 0 });
   builder.add('player', { name : 'Kif', score : 0 });

   // Build a content instance
   var content = builder.build();

   // Now that the content has been built, a topic can be added
with the metadata & initial value
   session.topics.add('games/yet-another-game', metadata,
content).then(function() {
       console.log('Topic was added with metadata and
content');
   });
```

```
});
```

The following example shows using the JavaScript Unified API to remove topics.

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {

    // 1. Like session.topics.add(), remove returns a promise,
 so we can chain together calls.
    session.topics.add('foo').then(session.topics.remove('foo'))
                              .then(function() {
                                 console.log('Removed topic
 foo');
                              }, function(reason) {
                                 console.log('Failed to remove
 topic foo: ', reason);
                              });

    // 2. Removing a topic will remove all topics underneath it.

    // Add a hierarchy of topics.
    var added =
 session.topics.add('a').then(session.topics.add('a/b'))

 .then(session.topics.add('a/b/c'))

 .then(session.topics.add('a/b/c/d'));

    // Wait until we've added all the topics
    added.then(session.topics.remove('a'))
         .then(function() {
            console.log('Removed all topics including & under
 "a"');
         });
});
```

Topics examples

The following example shows using the JavaScript Unified API to subscribe to topics.

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true
}).then(function(session) {

    // 1. Subscriptions are how sessions receive streams of data
 from the server.

    // When subscribing, a topic selector is used to select
 which topics to subscribe to. Topics do not need to exist
    // at the time of subscription - the server dynamically
 resolves subscriptions as topics are added or removed.

    // Subscribe to the "foo" topic with an inline callback
 function
```

```
    var subscription = session.subscribe('foo', function(update)
{
        // Log the new value whenever the 'foo' topic is updated
        // By default, we get a Buffer object which preserves
binary
        // data.
        console.log(update);
    });

    // Callbacks can also be registered after the subscription
has occurred
    subscription.on({
        update : function(value, topic) {
            console.log('Update for topic: ' + topic, value);
        },
        subscribe : function(details, topic) {
            console.log('Subscribed to topic: ' + topic);
        },
        unsubscribe : function(reason, topic) {
            console.log('Unsubscribed from topic:' + topic);
        }
    });

    // 2. Sessions may unsubscribe from any topic to stop
receiving data

    // Unsubscribe from the "foo" topic. Sessions do not need to
have previously been subscribed to the topics they are
    // unsubscribing from. Unsubscribing from a topic will
result in the 'unsubscribe' callback registered above being
    // called.
    session.unsubscribe('foo');

    // 3. Subscriptions / Unsubscriptions can select multiple
topics using Topic Selectors

    // Topic Selectors provide regex-like capabilities for
subscribing to topics. These are resolved dynamically, much
    // like subscribing to a single topic.
    var subscription2 = session.subscribe('?foo/.*/[a-z]');

    // 4. Subscriptions can use transformers to convert update
values

    // Subscribe to a topic and then convert all received values
to JSON. Transforming a subscription creates a new
    // subscription stream, rather than modifying the original.
    session.subscribe('bar').transform(JSON.parse).on('update',
function(value, topic) {
        console.log('Got JSON update for topic: ' + topic,
value);
    });

    // 5. Metadata can be used within transformers to parse data

    // Create a simple metadata instance
    var meta = new diffusion.metadata.RecordContent();

    // Add a single record/field
    meta.addRecord('record', {
        'field' : meta.string('some-value')
    });
```

```
    // Subscribe to a topic and transform with the metadata
    session.subscribe('baz').transform(meta).on('update',
 function(value) {
        console.log('Field value: ',
 value.get('record').get('field'));
    });
});
```

The following example shows views being used to present data from multiple topics as a single structure:

```
diffusion.connect({
    host        : 'localhost',
    port        : 8080,
    secure      : false,
    principal   : 'control',
    credentials : 'password'
}).then(function(session) {

    // Assuming a topic tree:
    //
    // scores
    //   |-- football
    //   |     |-- semi1
    //   |     |-- semi2
    //   |     |-- final
    //   |
    //   |-- tennis
    //         |-- semi1
    //         |-- semi2
    //         |-- final

    // Use a regular expression to create a view of the topics
 tracking the
    // scores during the finals for each sport.
    var view = session.view('?scores/.*/final');

    // Alternatively, we can use a topic set. Note that the
 topics do not need
    // to be under a common root, they may be anywhere within
 the topic tree.
    var view2 = session.view('#>scores/football/final////
>scores/tennis/final');

    // If any of the topics in the view change, display which
 topic changed
    // and its new value.
    view.on({
        update : function(value) {
            // Get and print the entire view structure.
            console.log('Update: ', JSON.stringify(value,
 undefined, 4));

            // Get individual topics. Returns a Buffer, which is
 automatically
            // converted to a String during concatenation,
 below.
            //
            // Note that the structure may not exist if the
 value has not been
            // updated.
```

```
            console.log('Football score: ' +
 value.scores.football.final);
            console.log('Tennis score  : ' +
 value.scores.tennis.final);

            // or ...
            // console.log('Football score: ' + value['scores']
['football']['final']);
        }
    });

    // The structure can also be accessed outside the update
 event.
    console.log('Football score: ' +
 view.get().scores.football.final);
});
```

TopicUpdateControl example

The following example shows using the JavaScript Unified API to send updates to topics.

```
diffusion.connect({
    host   : 'diffusion.example.com',
    port   : 443,
    secure : true,
    principal : 'control',
    credentials : 'password'
}).then(function(session) {
    // 1. A session may update any existing topic. Update values
 must be of the same type as the topic being updated.

    // Add a topic first with a string type
    session.topics.add('foo', '').then(function() {
        // Update the topic
        return session.topics.update('foo', 'hello');
    }).then(function() {
        // Update the topic again
        return session.topics.update('foo', 'world');
    });

    // 2. If using RecordContent metadata, update values are
 constructed from the metadata

    // Create a new metadata instance
    var meta = new diffusion.metadata.RecordContent();

    meta.addRecord('record', 1, {
        'field' : meta.integer()
    });

    // Create a builder to set values
    var builder = meta.builder();

    builder.add('record', {
        field : 123
    });

    // Update the topic with the new value
    session.topics.add('topic', '').then(function() {
        session.topics.update('topic', builder.build());
    });
```

```
});
```

C Unified API

The C API is provided in a number of zip files, each specific to an operating system.

The C library is provided with the Diffusion server installation in the `clients/c` folder.

For more information about using the C library on your preferred operating system, see Using the C Unified API on page 471.

Support

Table 58: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
C	• Linux: Red Hat and CentOS, version 6.5 and later (Tier 1 support) • Windows: Visual C Compiler 2013 (Tier 2 support) • OS X: (Tier 2 support)	• DPT, DPTS

Table 59: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
C	• Linux: Red Hat and CentOS, version 6.5 and later (Tier 1 support) • Windows: Visual C Compiler 2013 (Tier 2 support) • OS X: (Tier 2 support)	• DPT, DPTS

If you require libraries compiled on a different platform, this can be provided as an additional service by our Consulting Services team. Contact support@pushtechnology.com to discuss your requirements.

Features are sets of capabilities provided by the API.

Table 60: Capabilities provided by the API

Feature set	Supported capabilities
Authentication Control	Register an authentication handler that authenticates client connections.
Messaging	Send messages through a topic path to the message handler of that topic path

Feature set	Supported capabilities
	Receive messages sent through a topic path
Messaging Control	Receive messages sent to the message handler of a topic path
Pings	Ping the Diffusion server
Security	Change the security principal and credentials associated with the client session
System Authentication Control	Define users that are authenticated by the system authentication handler.
Topic Control	Create topics Delete topics
Topics	Subscribe to topics (excluding paged topics) Receive streaming updates on subscribed topics (excluding paged topics) Fetch the current state of a topic (excluding paged topics) Unsubscribe from topics (excluding paged topics)
Topic Update Control	Register as a topic update source Update topics exclusively (excluding paged topics)

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 61: Capabilities provided by the library

Library capability	Supported
Connect with multiple server details	✗
Transport cascading	✗
Connect through load balancer	✓
Reconnect	✓ PARTIAL

Getting started with C

The example demonstrates an empty C client that you can base your clients on.

Before you begin

The C client libraries rely on the following dependencies:

- APR version 1.5 with APR-util library

  For more information, see http://apr.apache.org
- Perl Compatible Regular Expressions (PCRE) library, version 8.3

  For more information, see http://pcre.org
- OpenSSL library, version 1.0.2a

  For more information, see https://www.openssl.org

Ensure that these libraries are available on your development system.

For Linux or OS X, you can download them through your operating system's package manager.

For Windows, Push Technology provides custom builds of these libraries built with the same compiler and on the same architecture as the C Diffusion client libraries.

Procedure

1. Create a C file that contains the client `main` method. For example, `example.c`.

```
#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://diffusion.example.com:80"},
        END_OF_ARG_OPTS
};

int
main(int argc, char **argv)
{
        // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") !=
 NULL) {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");

        // A SESSION_LISTENER_T holds callbacks to inform the
 client
        // about changes to the state. Used here for
 informational
        // purposes only.
        SESSION_LISTENER_T session_listener;
        session_listener.on_state_changed =
 &on_session_state_changed;
```

```
        session_listener.on_state_error =
&on_session_state_error;

        // Create a session with the Diffusion server.
        DIFFUSION_ERROR_T error;
        SESSION_T *session = NULL;
        session = session_create(url, NULL, NULL,
&session_listener, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create
session\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

// Set up the features you want to use with your client
session.

        // Start the session.
        if(! session_start(session, &error)) {
                fprintf(stderr, "ERR: Failed to start session:
%s\n", error.message);
                return 1;
        }

        // Use the session and features.
        // ...

        // Close the session with the Diffusion server
        session_close(session, &error);

        return 0;
}
```

a) Include the `diffusion.h` header file.

   This file pulls in the other required header files.

b) In the `main` method, use the `session_create` function to create and configure the session.

   The Diffusion server connection URL is the only mandatory parameter for this function. All other parameters can take a `NULL` value.

   Change the URL from that provided in the example to the URL of the Diffusion server.

c) Do any setup on the session as required. For example, subscribe to topics.

d) Use the `session_start` function to start the session.

e) Use the session to perform your client actions.

   All functions that use the client features take the `session` as a parameter in addition to feature-specific arguments and handlers.

f) Use the `session_close` function to disconnect the client from the server.

2. Review any operating system specific requirements for compiling your client.

For more information, see Using the C Unified API on page 471.

3. Create a makefile to build your client.

For example, on Linux:

```
CC  = gcc
AR  = ar
CFLAGS  = -c -g -Wall -I../include -I/usr/include/apr-1
LDFLAGS  = -L../lib -lapr-1 -laprutil-1 -lpcre -lssl -
ldiffusion
```

```
ARFLAGS  =
SOURCES  = example.c
OBJECTS  = $(SOURCES:.c=.o)
TARGETS  = example

all:  $(SOURCES) $(TARGETS)

.c.o:
  $(CC) $(CFLAGS) $< -o $@

clean:
  rm -f $(OBJECTS) $(TARGETS) core a.out
```

a) Link against one of the Diffusion libraries:
   - `libdiffusion.so` for dynamic binding on Linux
   - `libdiffusion.a` for static binding on Linux or OS X
   - `uci.lib` on Windows

b) Link against all of the following libraries:
   - apr
   - apr-util
   - pcre
   - openssl

Using the C Unified API

There are features, issues, and considerations that are specific to clients that are implemented using the C Unified API.

- On Linux on page 471
- On Windows on page 472
- On OS X on page 472
- Defining the structure of record topic data using XML on page 472
- Threading model on page 473

On Linux

The C libraries are provided compiled for 64-bit Linux in the file `diffusion-c-`*`version`*`.zip`. A dynamic library, `libdiffusion.so`, and a static library, `libdiffusion.a`, are available. These libraries are supported on Red Hat and CentOS version 6.5 and later.

To use the C Unified API on Linux ensure that the following dependencies are available on your development system:

- APR version 1.5 with APR-util library

  For more information, see http://apr.apache.org
- Perl Compatible Regular Expressions (PCRE) library, version 8.3

  For more information, see http://pcre.org
- OpenSSL library, version 1.0.2a

  For more information, see https://www.openssl.org

You can download these dependencies through your operating system's package manager.

On Windows

The C library is provided as a static library compiled for 32-bit and 64-bit Windows in the file `diffusion-c-windows-`*`version`*`.zip`. This static library, `uci.lib`, is compiled with Visual C Compiler 2013 (version 120), which is shipped by default with Microsoft Visual Studio 2013. You must use this version of Visual C Compiler or later. Earlier versions are not supported.

Other Windows compilers, such as Clang and GCC, are not supported.

When compiling with Visual C Compiler 2013, define `/D WIN32` in the compiler settings.

The `diffusion-c-windows-`*`version`*`.zip` file contains custom builds of the following dynamic libraries:

- APR version 1.5 with APR-util library

  For more information, see http://apr.apache.org
- Perl Compatible Regular Expressions (PCRE) library, version 8.3

  For more information, see http://pcre.org
- OpenSSL library, version 1.0.2a

  For more information, see https://www.openssl.org

We recommend that you use these provided binaries instead of building your own.

On OS X

The C library is provided as a static library, `libdiffusion.a`, compiled for 64-bit OS X in the file `diffusion-c-osx-`*`version`*`.zip`.

To use the C Unified API on OS X ensure that the following dependencies are available on your development system:

- APR version 1.5 with APR-util library

  For more information, see http://apr.apache.org
- Perl Compatible Regular Expressions (PCRE) library, version 8.3

  For more information, see http://pcre.org
- OpenSSL library, version 1.0.2a

  For more information, see https://www.openssl.org

You can download these dependencies using `brew`.

Defining the structure of record topic data using XML

Data on record topics can be structured using metadata. Other Diffusion APIs provide builder methods you can use to define the metadata structure. The C Unified API uses XML to define the structure of a record topic's metadata.

The following schema describes the structure of that XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/
XMLSchema">

  <xs:element name="field" type="field"/>

  <xs:element name="message" type="message"/>

  <xs:element name="record" type="record"/>

  <xs:complexType name="record">
```

```
    <xs:complexContent>
      <xs:extension base="node">
        <xs:sequence>
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="record"/>
            <xs:element ref="field"/>
          </xs:choice>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="node">
    <xs:sequence/>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="multiplicity" type="xs:string"/>
  </xs:complexType>

  <xs:complexType name="field">
    <xs:complexContent>
      <xs:extension base="node">
        <xs:sequence/>
        <xs:attribute name="type" type="dataType"
 use="required"/>
        <xs:attribute name="default" type="xs:string"/>
        <xs:attribute name="scale" type="xs:integer"/>
        <xs:attribute name="allowsEmpty" type="xs:boolean"/>
        <xs:attribute name="customFieldHandlerClassName"
 type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="message">
    <xs:complexContent>
      <xs:extension base="record">
        <xs:sequence/>
        <xs:attribute name="topicDataType" type="topicDataType"/
>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:simpleType name="dataType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="integerString"/>
      <xs:enumeration value="string"/>
      <xs:enumeration value="customString"/>
      <xs:enumeration value="decimalString"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="topicDataType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="record"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Threading model

The C Unified API is thread-safe.

Internally, it creates the following threads for each connection to the Diffusion server:

Write thread

When the client wants to send any data to the Diffusion server, the data is first written to a queue. The write thread waits for data to be available on this queue and sends it in the background to the Diffusion server.

Read thread

The read thread waits for data to be delivered from the Diffusion server, processes that data, and dispatches the data to user-supplied callbacks as appropriate. The callbacks are invoked synchronously. If the callbacks run for a significant amount of time, consider handing off the processing in your callback to another thread.

Related Links

Metadata in the Unified API on page 173
Metadata is available in the Unified API. It defines how the bytes in your content are formatted.

C examples

Examples that use the C Unified API.

- AuthenticationControl example on page 474
- Messaging example on page 478
- Pings example on page 480
- Security example on page 481
- SystemAuthenticationControl example on page 483
- TopicControl example on page 485
- Topics example on page 490

AuthenticationControl example

The following example shows using the C Unified API to register a control authentication handler with the Diffusion server. The example also includes a simple or empty authentication handler.

The name by which the control authentication handler is registered must be configured in the `Server.xml` configuration file of the Diffusion server for the control authentication handler to be called to handle authentication requests.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"
#include "conversation.h"

struct user_credentials_s {
        const char *username;
        const char *password;
};

static const struct user_credentials_s USERS[] = {
        { "fish", "chips" },
        { "ham", "eggs" },
        { NULL, NULL }
};
```

```
ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'n', "name", "Name under which to register the
 authorisation handler", ARG_OPTIONAL, ARG_HAS_VALUE, "c-auth-
handler"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
 * When the authentication service has been registered, this
 function will be
 * called.
 */
static int
on_registration(SESSION_T *session, void *context)
{
        printf("Registered authentication handler\n");
        return HANDLER_SUCCESS;
}

/**
 * When the authentication service has be deregistered, this
 function will be
 * called.
 */
static int
on_deregistration(SESSION_T *session, void *context)
{
        printf("Deregistered authentication handler\n");
        return HANDLER_SUCCESS;
}

/**
 * This is the function that is called when authentication has
 been delegated
 * from Diffusion.
 *
 * The response may return one of three values via the response
 parameter:
 * ALLOW:   The user is authenticated.
 * ALLOW_WITH_RESULT: The user is authenticated, and additional
 roles are
 *                    to be applied to the user.
 * DENY:    The user is NOT authenticated.
 * ABSTAIN: Allow another handler to make the decision.
 *
 * The handler should return HANDLER_SUCCESS in all cases,
 unless an actual
 * error occurs during the authentication process (in which
 case,
 * HANDLER_FAILURE is appropriate).
 */
static int
on_authentication(SESSION_T *session,
                  const SVC_AUTHENTICATION_REQUEST_T *request,
                  SVC_AUTHENTICATION_RESPONSE_T *response,
                  void *context)
```

```
{
        int auth_decided = 0;

        // No credentials, or not password type. We're not an
 authority for
        // this type of authentication so abstain in case some
 other registered
        // authentication handler can deal with the request.
        if(request->credentials == NULL) {
                printf("No credentials specified, abstaining
\n");
                response->value = AUTHENTICATION_ABSTAIN;
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }
        if(request->credentials->type != PLAIN_PASSWORD) {
                printf("Credentials are not PLAIN_PASSWORD,
 abstaining\n");
                response->value = AUTHENTICATION_ABSTAIN;
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }

        printf("principal = %s\n", request->principal);
        printf("credentials = %*s\n",
               (int)request->credentials->data->len,
               request->credentials->data->data);

        if(request->principal == NULL || strlen(request-
>principal) == 0) {
                printf("Denying anonymous connection (no
 principal)\n");
                response->value = AUTHENTICATION_DENY; // Deny
 anon connections
                auth_decided = 1;
                return HANDLER_SUCCESS;
        }

        char *password = malloc(request->credentials->data->len
 + 1);
        memmove(password, request->credentials->data->data,
 request->credentials->data->len);
        password[request->credentials->data->len] = '\0';

        int i = 0;
        while(USERS[i].username != NULL) {

                printf("Checking username %s vs %s\n", request-
>principal, USERS[i].username);
                printf("     and password %s vs %s\n", password,
 USERS[i].password);

                if(strcmp(USERS[i].username, request->principal)
 == 0 &&
                   strcmp(USERS[i].password, password) == 0) {

                        puts("Allowed");
                        response->value = AUTHENTICATION_ALLOW;
                        auth_decided = 1;
                        break;
                }
                i++;
```

```
        }

        free(password);

        if(auth_decided == 0) {
                puts("Abstained");
                response->value = AUTHENTICATION_ABSTAIN;
                auth_decided = 1;
        }

        return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if (options == NULL || hash_get(options, "help") !=
 NULL) {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        char *name = hash_get(options, "name");
        char *principal = hash_get(options, "principal");
        char *credentials = hash_get(options, "credentials");

        // Create a session with Diffusion.
        puts("Creating session");
        DIFFUSION_ERROR_T error;
        SESSION_T *session = session_create(url,
                                            principal,
                                            credentials !=
 NULL ? credentials_create_password(credentials) : NULL,
                                            NULL, NULL,
                                            &error);
        if (session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        // Provide a set (via a hash map containing keys and
 NULL values)
        // to indicate what information about the connecting
 client that we'd
        // like Diffusion to send us.
        HASH_T *detail_set = hash_new(5);
        char buf[2];
        sprintf(buf, "%d", SESSION_DETAIL_SUMMARY);
        hash_add(detail_set, strdup(buf), NULL);
        sprintf(buf, "%d", SESSION_DETAIL_LOCATION);
        hash_add(detail_set, strdup(buf), NULL);
        sprintf(buf, "%d", SESSION_DETAIL_CONNECTOR_NAME);
        hash_add(detail_set, strdup(buf), NULL);

        AUTHENTICATION_REGISTRATION_PARAMS_T
 auth_registration_params = {
                .name = name,
                .detail_set = detail_set,
                .on_registration = on_registration,
```

```
                .authentication_handlers.on_authentication =
 on_authentication
        };

        // Register the authentication handler.
        puts("Sending registration request");
        SVC_AUTHENTICATION_REGISTER_REQUEST_T *reg_request =
                authentication_register(session,
 auth_registration_params);

        // Wait a while before moving on to deregistration.
        sleep(10);

        AUTHENTICATION_DEREGISTRATION_PARAMS_T
 auth_deregistration_params = {
                .on_deregistration = on_deregistration,
                .original_request = reg_request
        };

        printf("Deregistering authentication handler\n");
        authentication_deregister(session,
 auth_deregistration_params);

        // Never exit
        while (1) {
                sleep(10);
        }

        // Not called, but this is the way you would gracefully
 terminate the
        // connection with Diffusion.
        session_close(session, &error);

        return(EXIT_SUCCESS);
}
```

Messaging example

The following example shows using the C Unified API to send a message to a topic path.

```
#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'t', "topic", "Topic name", ARG_REQUIRED,
 ARG_HAS_VALUE, "echo"},
        {'d', "data", "Data to send", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
```

```
 * Callback invoked when/if a message is published on the topic
 that the
 * client is writing to.
 */
static int
topic_handler(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
        printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
 return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }
        char *topic = hash_get(options, "topic");

 // For subscription, we create a selector which specifies the
 topic
 // we're sending to *only*.
        char *topic_selector = malloc(1 + strlen(topic) + 1);
        topic_selector[0] = '>';
        strcpy(&topic_selector[1], topic);

 // Create a session with Diffusion.
        SESSION_T *session = NULL;
        DIFFUSION_ERROR_T error;
        session = session_create(url, principal, credentials,
 NULL, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

 // Install a subscription handler.
 SUBSCRIPTION_HANDLERS_T *handlers = calloc(1,
 sizeof(SUBSCRIPTION_HANDLERS_T));
 handlers->on_topic_message = &topic_handler;
 subscribe(session, (SUBSCRIPTION_PARAMS_T) { .topic_selector =
 topic_selector, .on_topic_message = topic_handler });

 // Create a message.
        char *data = hash_get(options, "data");
```

```
        LIST_T *headers = list_create();
        BUF_T *payload = buf_create();
        buf_write_bytes(payload, data, strlen(data));

        MESSAGE_T msg = {
                .type = MESSAGE_TYPE_TOPIC_LOAD,
                .headers = headers,
                .payload = payload
        };

 // Send the message.
        SEND_MSG_PARAMS_T params = {
                .topic_path = hash_get(options, "topic"),
                .msg = msg,
                .priority = MESSAGE_PRIORITY_MEDIUM
        };
        send_msg(session, params);

 // Wait a few seconds; send_msg is asynchronous and it we don't
 want
 // to kill the session until the message has been given chance
 to be
 // sent. We also want to wait for any possible responses.
        sleep(5);

 // Politely close the client connection.
        session_close(session, &error);

        return 0;
}
```

Pings example

The following example shows using the C Unified API to ping Diffusion from a client.

```
#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        END_OF_ARG_OPTS
};

/**
 * Callback for displaying the receipts of a ping response.
 */
static int
on_ping_response(SESSION_T *session, const SVC_PING_RESPONSE_T
 *response, void *context)
{
        printf("Received ping response\n");
        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
```

```
        // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");

        // Create a session with the Diffusion server.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, NULL, NULL, NULL, NULL,
 &error);
        if(session == NULL) {
                fprintf(stderr, "Failed to create session: %s
\n", error.message);
                return 1;
        }

        // Define a ping response handler
        PING_PARAMS_T ping_params = {
                .on_ping_response = on_ping_response
        };

        // Send 5 pings at 1 sec intervals
        int i;
        for(i = 0; i < 5; i++) {
                ping(session, ping_params);
                sleep(1);
        }

        // Gracefully close the client session.
        session_close(session, &error);

        return 0;
}
```

Security example

The following example shows using the C Unified API to change the client principal and credentials for a client session.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};
```

```
/*
 * Callback to display that the change_principal() request has
 been processed
 * by Diffusion.
 */
static int
on_change_principal(SESSION_T *session, void *context)
{
        printf("on_change_principal\n");
        return HANDLER_SUCCESS;
}

int
main(int argc, char** argv)
{
        // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");

        // Create a session with Diffusion, with no principal or
 credentials.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, NULL, NULL, NULL, NULL,
 &error);
        if(session == NULL) {
                fprintf(stderr, "Failed to create session: %s
\n", error.message);
                return 1;
        }

        // Wait for a couple of seconds.
        sleep(2);

        puts("Changing credentials");

        CREDENTIALS_T *credentials =
 credentials_create_password("chips");

        // Specify callbacks for the change_principal request.
        CHANGE_PRINCIPAL_PARAMS_T params = {
                .principal = "fish",
                .credentials = credentials,
                .on_change_principal = on_change_principal
        };

        // Do the change.
        change_principal(session, params);

        // Wait for a couple more seconds.
        sleep(2);

        puts("Closing session");

        // Gracefully close the connection.
        session_close(session, &error);
```

```
        return 0;
}
```

SystemAuthenticationControl example

The following example shows using the C Unified API to update the contents of the system authentication store on the Diffusion server.

```
#include <stdio.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "service/svc-system-auth-control.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

int
on_get_system_authentication_store(SESSION_T *session,
                                   const
 SYSTEM_AUTHENTICATION_STORE_T store,
                                   void *context)
{
        puts("on_get_system_authentication_store()");

        printf("Got %ld principals\n", store.system_principals-
>size);

        char **names = get_principal_names(store);
        for(char **name = names; *name != NULL; name++) {
                printf("Principal: %s\n", *name);

                char **roles = get_roles_for_principal(store,
 *name);
                for(char **role = roles; *role != NULL; role++)
 {
                    printf("  |- Role: %s\n", *role);
                }
                free(roles);
        }
        free(names);

        switch(store.anonymous_connection_action) {
        case ANONYMOUS_CONNECTION_ACTION_ALLOW:
                puts("Allow anonymous connections");
                break;
```

```
        case ANONYMOUS_CONNECTION_ACTION_DENY:
                puts("Deny anonymous connections");
                break;
        case ANONYMOUS_CONNECTION_ACTION_ABSTAIN:
                puts("Abstain from making anonymous connection
 decision");
                break;
        }

        puts("Anonymous connection roles:");
        char **roles = get_anonymous_roles(store);
        for(char **role = roles; *role != NULL; role++) {
                printf("  |- Role: %s\n", *role);
        }
        free(roles);

        apr_thread_cond_broadcast(cond);

        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
        // Standard command line parsing.
        const HASH_T *options = parse_cmdline(argc, argv,
 arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        const char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }

        // Setup for condition variable
        apr_initialize();
        apr_pool_create(&pool, NULL);
        apr_thread_mutex_create(&mutex,
 APR_THREAD_MUTEX_DEFAULT, pool);
        apr_thread_cond_create(&cond, pool);

        // Setup for session
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url, principal, credentials,
 NULL, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        const GET_SYSTEM_AUTHENTICATION_STORE_PARAMS_T params =
 {
```

```
                .on_get = on_get_system_authentication_store
        };

        apr_thread_mutex_lock(mutex);

        get_system_authentication_store(session, params);

        apr_thread_cond_wait(cond, mutex);
        apr_thread_mutex_unlock(mutex);

        return 0;
}
```

TopicControl example

The following example shows using the C Unified API to create topics.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <apr.h>
#include <apr_thread_mutex.h>
#include <apr_thread_cond.h>

#include "diffusion.h"
#include "args.h"
#include "service/svc-topic-control.h"
#include "utils.h"

apr_pool_t *pool = NULL;
apr_thread_mutex_t *mutex = NULL;
apr_thread_cond_t *cond = NULL;

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'p', "principal", "Principal (username) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

// Various handlers which are common to all add_topic()
 functions.
static int
on_topic_added(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 puts("on_topic_added");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_add_failed(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 printf("on_topic_add_failed: %d\n", response->response_code);
 apr_thread_cond_broadcast(cond);
```

```
 return HANDLER_SUCCESS;
}

static int
on_topic_add_discard(SESSION_T *session, const
 SVC_ADD_TOPIC_RESPONSE_T *response, void *context)
{
 puts("on_topic_add_discard");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_removed(SESSION_T *session, const
 SVC_REMOVE_TOPICS_RESPONSE_T *response, void *context)
{
 puts("on_topic_removed");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}

static int
on_topic_remove_discard(SESSION_T *session, const
 SVC_REMOVE_TOPICS_RESPONSE_T *response, void *context)
{
 puts("on_topic_remove_discard");
 apr_thread_cond_broadcast(cond);
 return HANDLER_SUCCESS;
}
/*
 *
 */
int main(int argc, char** argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        const char *principal = hash_get(options, "principal");
        CREDENTIALS_T *credentials = NULL;
        const char *password = hash_get(options, "credentials");
        if(password != NULL) {
                credentials =
 credentials_create_password(password);
        }

 // Setup for condition variable
 apr_initialize();
 apr_pool_create(&pool, NULL);
 apr_thread_mutex_create(&mutex, APR_THREAD_MUTEX_DEFAULT,
 pool);
 apr_thread_cond_create(&cond, pool);

 // Setup for session
 SESSION_T *session;
 DIFFUSION_ERROR_T error;
 session = session_create(url, principal, credentials, NULL,
 NULL, &error);
```

```
 if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

        // Common params for all add_topic() functions.
        ADD_TOPIC_PARAMS_T common_params = {
                .on_topic_added = on_topic_added,
                .on_topic_add_failed = on_topic_add_failed,
                .on_discard = on_topic_add_discard
        };

 // Stateless topic details
 TOPIC_DETAILS_T *topic_details =
 create_topic_details_stateless();
        ADD_TOPIC_PARAMS_T stateless_params = common_params;
        stateless_params.topic_path = "stateless";
        stateless_params.details = topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, stateless_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value string data, no default data.
 TOPIC_DETAILS_T *string_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_STRING);
        ADD_TOPIC_PARAMS_T string_params = common_params;
        string_params.topic_path = "string";
        string_params.details = string_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, string_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value string data, with default data.
        ADD_TOPIC_PARAMS_T string_data_params = common_params;
        string_data_params.topic_path = "string-data";
        string_data_params.details = string_topic_details;
 BUF_T *sample_data_buf = buf_create();
 buf_write_string(sample_data_buf, "Hello, world");
        string_data_params.content =
 content_create(CONTENT_ENCODING_NONE, sample_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, string_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value integer data.
        TOPIC_DETAILS_T *integer_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_INTEGER_STRING);
 integer_topic_details-
>topic_details_params.integer.default_value = 999;

        ADD_TOPIC_PARAMS_T integer_params = common_params;
        integer_params.topic_path = "integer";
        integer_params.details = integer_topic_details;

 apr_thread_mutex_lock(mutex);
```

```
 add_topic(session, integer_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value integer data with content.
        ADD_TOPIC_PARAMS_T integer_data_params = common_params;
        integer_data_params.topic_path = "integer-data";
        integer_data_params.details = integer_topic_details;
 BUF_T *integer_data_buf = buf_create();
 buf_sprintf(integer_data_buf, "%d", 123);
        integer_data_params.content =
 content_create(CONTENT_ENCODING_NONE, integer_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, integer_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value decimal data.
 TOPIC_DETAILS_T *decimal_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_DECIMAL_STRING);
 decimal_topic_details-
>topic_details_params.decimal.default_value = 123.456;
 decimal_topic_details->topic_details_params.decimal.scale = 4;

        ADD_TOPIC_PARAMS_T decimal_params = common_params;
        decimal_params.topic_path = "decimal";
        decimal_params.details = decimal_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, decimal_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Single value decimal data with content.
        ADD_TOPIC_PARAMS_T decimal_data_params = common_params;
        decimal_data_params.topic_path = "decimal-data";
        decimal_data_params.details = decimal_topic_details;
 BUF_T *decimal_data_buf = buf_create();
 buf_sprintf(decimal_data_buf, "%f", 987.654);
 decimal_data_params.content =
 content_create(CONTENT_ENCODING_NONE, decimal_data_buf);

 apr_thread_mutex_lock(mutex);
 add_topic(session, decimal_data_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Manually specify schema for single value string topic data.
 BUF_T *manual_schema = buf_create();
 buf_write_string(manual_schema,
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>
\n");
 buf_write_string(manual_schema,
  "<field name=\"x\" type=\"string\" default=\"xyzzy\"
 allowsEmpty=\"true\"/>");
 TOPIC_DETAILS_T *manual_topic_details =
 create_topic_details_single_value(M_DATA_TYPE_STRING);
 manual_topic_details->user_defined_schema = manual_schema;

        ADD_TOPIC_PARAMS_T string_manual_params = common_params;
        string_manual_params.topic_path = "string-manual";
        string_manual_params.details = manual_topic_details;
```

```
 apr_thread_mutex_lock(mutex);
 add_topic(session, string_manual_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Simple record/field structure, created by manually
 specifying the
 // schema.
 BUF_T *record_schema = buf_create();
 buf_write_string(record_schema,
  "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?
>");
 buf_write_string(record_schema,
  "<message topicDataType=\"record\" name=\"MyContent\">");
 buf_write_string(record_schema,
  "<record name=\"Record1\">");
 buf_write_string(record_schema,
  "<field type=\"string\" default=\"\" allowsEmpty=\"true\"
 name=\"Field1\"/>");
 buf_write_string(record_schema,
  "<field type=\"integerString\" default=\"0\" allowsEmpty=
\"false\" name=\"Field2\"/>");
 buf_write_string(record_schema,
  "<field type=\"decimalString\" default=\"0.00\" scale=\"2\"
 allowsEmpty=\"false\" name=\"Field3\"/>");
 buf_write_string(record_schema,
  "</record>");
 buf_write_string(record_schema,
  "</message>");
 TOPIC_DETAILS_T *record_topic_details =
 create_topic_details_record();
 record_topic_details->user_defined_schema = record_schema;

        ADD_TOPIC_PARAMS_T record_params = common_params;
        record_params.topic_path = "record";
        record_params.details = record_topic_details;

 apr_thread_mutex_lock(mutex);
 add_topic(session, record_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 // Remove topic tests
 puts("Adding topics remove_me/1 and remove_me/2");
 apr_thread_mutex_lock(mutex);
        ADD_TOPIC_PARAMS_T topic_params = common_params;
        topic_params.details = topic_details;

        topic_params.topic_path = "remove_me/1";
 add_topic(session, topic_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

        topic_params.topic_path = "remove_me/2";
 add_topic(session, topic_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 puts("Removing topics in 5 seconds...");
 sleep(5);

        REMOVE_TOPICS_PARAMS_T remove_params = {
```

```
                .on_removed = on_topic_removed,
                .on_discard = on_topic_remove_discard,
                .topic_selector = ">remove_me"
        };

 apr_thread_mutex_lock(mutex);
 remove_topics(session, remove_params);
 apr_thread_cond_wait(cond, mutex);
 apr_thread_mutex_unlock(mutex);

 return(EXIT_SUCCESS);
}
```

Topics example

The following example shows using the C Unified API to subscribe to topics and receive the topic content.

```
/*
 * This is a sample client which connects to Diffusion v5 and
 subscribes to
 * topics using a user-specified selector. Any messages received
 on those topics
 * are then displayed to standard output.
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/*
 * This callback is used when the session state changes, e.g.
 when a session
 * moves from a "connecting" to a "connected" state, or from
 "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session,
                         const SESSION_STATE_T old_state,
                         const SESSION_STATE_T new_state)
{
        printf("Session state changed from %s (%d) to %s
 (%d)\n",
                session_state_as_string(old_state), old_state,
                session_state_as_string(new_state), new_state);
}

/*
 * When a subscribed message is received, this callback is
 invoked.
```

```
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
        printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
        return HANDLER_SUCCESS;
}

/*
 * This callback is fired when Diffusion responds to say that a
 topic
 * subscription request has been received and processed.
 */
static int
on_subscribe(SESSION_T *session, void *context_data)
{
        printf("on_subscribe\n");
        return HANDLER_SUCCESS;
}

/*
 * This is callback is for when Diffusion response to an
 unsubscription
 * request to a topic, and only indicates that the request has
 been received.
 */
static int
on_unsubscribe(SESSION_T *session, void *context_data)
{
        printf("on_unsubscribe\n");
        return HANDLER_SUCCESS;
}

/*
 * Publishers and control clients may choose to subscribe any
 other client to
 * a topic of their choice at any time. We register this
 callback to capture
 * messages from these topics and display them.
 */
static int
on_unexpected_topic_message(SESSION_T *session, const
 TOPIC_MESSAGE_T *msg)
{
        printf("Received a message for a topic we didn't
 subscribe to (%s)\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);
        return HANDLER_SUCCESS;
}

/*
 * We use this callback when Diffusion notifies us that we've
 been subscribed
 * to a topic. Note that this could be called for topics that we
 haven't
 * explicitly subscribed to - other control clients or
 publishers may ask to
 * subscribe us to a topic.
 */
static int
```

```
on_notify_subscription(SESSION_T *session, const
 SVC_NOTIFY_SUBSCRIPTION_REQUEST_T *request, void *context)
{
        printf("on_notify_subscription: %d: \"%s\"\n",
               request->topic_info.topic_id,
               request->topic_info.topic_path);
        return HANDLER_SUCCESS;
}

/*
 * This callback is used when we receive notification that this
 client has been
 * unsubscribed from a specific topic. Causes of the
 unsubscription are the same
 * as those for subscription.
 */
static int
on_notify_unsubscription(SESSION_T *session, const
 SVC_NOTIFY_UNSUBSCRIPTION_REQUEST_T *request, void *context)
{
        printf("on_notify_unsubscription: %d, reason: %d\n",
               request->topic_id,
               request->reason);
        return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
        // Standard command line parsing
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        char *topic = hash_get(options, "topic_selector");

        // A SESSION_LISTENER_T holds callbacks to inform the
 client
        // about changes to the state. Used here for
 informational
        // purposes only.
        SESSION_LISTENER_T session_listener;
        session_listener.on_state_changed =
 &on_session_state_changed;

        // Creating a session requires at least a URL. Creating
 a session
        // initiates a connection with Diffusion.
        DIFFUSION_ERROR_T error;
        SESSION_T *session = NULL;
        session = session_create(url, NULL, NULL,
 &session_listener, NULL, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }
```

```
        // When issuing commands to Diffusion (in this case,
 subscribe to
        // a topic), it's typical that more than one message may
 be
        // received in response and a handler can be installed
 for each
        // message type. In the case of subscription, we can
 install
        // handlers for:
        // 1. The topic message data (on_topic_message).
        // 2. Notification that the subscription has been
 received (on_subscribe).
        // 3. Topic details (on_topic_details).

        notify_subscription_register(session,
 (NOTIFY_SUBSCRIPTION_PARAMS_T) { .on_notify_subscription =
 on_notify_subscription });
        notify_unsubscription_register(session,
 (NOTIFY_UNSUBSCRIPTION_PARAMS_T) { .on_notify_unsubscription =
 on_notify_unsubscription });

        subscribe(session, (SUBSCRIPTION_PARAMS_T)
 { .topic_selector = topic, .on_topic_message =
 on_topic_message, .on_subscribe = on_subscribe });

        // Install a global topic handler to capture messages
 for topics we haven't
        // explicitly subscribed to, and therefore don't have a
 specific handler for.
        session->global_topic_handler =
 on_unexpected_topic_message;

        // Keep receiving messages for 5 seconds.
        sleep(5);

        // Unsubscribe from the topic
        unsubscribe(session, (UNSUBSCRIPTION_PARAMS_T)
 {.topic_selector = topic, .on_unsubscribe = on_unsubscribe} );

        // Wait for any unsubscription notifications to be
 received.
        sleep(5);

        // Politely tell Diffusion we're closing down.
        session_close(session, &error);

        return 0;
}
```

The following example shows using the C Unified API to fetch the current state of a topic without subscribing to the topic.

```
/*
 * This is a sample client which connects to Diffusion v5 and
 demonstrates
 * the following features:
 *
 * 1. Fetch topic state using a user-specified topic selector.
 * 2. Connect to Diffusion with a username and password.
 * 3. Automatic retry of a connection if unable to connect at
 the first
 *    attempt.
```

```
 */

#include <stdio.h>
#include <unistd.h>

#include "diffusion.h"
#include "args.h"

extern void topic_message_debug();

ARG_OPTS_T arg_opts[] = {
        ARG_OPTS_HELP,
        {'u', "url", "Diffusion server URL", ARG_OPTIONAL,
 ARG_HAS_VALUE, "dpt://localhost:8081"},
        {'t', "topic_selector", "Topic selector", ARG_REQUIRED,
 ARG_HAS_VALUE, NULL},
        {'r', "retries", "Number of connection retries",
 ARG_OPTIONAL, ARG_HAS_VALUE, "3"},
        {'d', "retry_delay", "Delay (in ms) between connection
 attempts", ARG_OPTIONAL, ARG_HAS_VALUE, "1000"},
 {'p', "principal", "Principal (username) for the connection",
 ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
 {'c', "credentials", "Credentials (password) for the
 connection", ARG_OPTIONAL, ARG_HAS_VALUE, NULL},
        END_OF_ARG_OPTS
};

/**
 * This callback is used when the session state changes, e.g.
 when a session
 * moves from a "connecting" to a "connected" state, or from
 "connected" to
 * "closed".
 */
static void
on_session_state_changed(SESSION_T *session, const
 SESSION_STATE_T old_state, const SESSION_STATE_T new_state)
{
        printf("Session state changed from %s (%d) to %s
 (%d)\n",
                session_state_as_string(old_state), old_state,
                session_state_as_string(new_state), new_state);
        if(new_state == CONNECTED_ACTIVE) {
                 printf("Session ID=%s\n",
 session_id_to_string(session->id));
        }
}

/**
 * This callback is invoked when Diffusion acknowledges that it
 has received
 * the fetch request. It does not indicate that there will be
 any subsequent
 * messages; see on_topic_message() and
 on_fetch_status_message() for that.
 */
static int
on_fetch(SESSION_T *session, void *context)
{
 puts("Fetch acknowledged by server");
 return HANDLER_SUCCESS;
}
```

```
/**
 * This callback is invoked when all messages for a topic
 selector have
 * been received, or there was some kind of server-side error
 during the
 * fetch processing.
 */
static int
on_fetch_status_message(SESSION_T *session,
                        const SVC_FETCH_STATUS_RESPONSE_T
 *status,
                        void *context)
{
 switch(status->status_flag) {
 case DIFFUSION_TRUE:
  puts("Fetch succeeded");
  break; //exit(0);
 case DIFFUSION_FALSE:
  puts("Fetch failed");
  break; //exit(1);
 default:
  printf("Unknown fetch status: %d\n", status->status_flag);
  break;
 }

 return HANDLER_SUCCESS;
}

/**
 * When a fetched message is received, this callback in invoked.
 */
static int
on_topic_message(SESSION_T *session, const TOPIC_MESSAGE_T *msg)
{
 printf("Received message for topic %s\n", msg->name);
        printf("Payload: %.*s\n", (int)msg->payload->len, msg-
>payload->data);

#ifdef DEBUG
         topic_message_debug(response->payload);
#endif

 return HANDLER_SUCCESS;
}

int
main(int argc, char **argv)
{
 // Standard command line parsing.
        HASH_T *options = parse_cmdline(argc, argv, arg_opts);
        if(options == NULL || hash_get(options, "help") != NULL)
 {
                show_usage(argc, argv, arg_opts);
                return 1;
        }

        char *url = hash_get(options, "url");
        char *topic = hash_get(options, "topic_selector");
        int retries = atoi(hash_get(options, "retries"));
        long retry_delay = atol(hash_get(options,
 "retry_delay"));

 // A SESSION_LISTENER_T holds callbacks to inform the client
```

```
        // about changes to the state. Used here for
 informational
        // purposes only.
        SESSION_LISTENER_T foo_listener;
        foo_listener.on_state_changed =
 &on_session_state_changed;

 // The client-side API can automatically keep retrying to
 connect
 // to the Diffusion server if it's not immediately available.
        SESSION_FAILOVER_STRATEGY_T failover_strategy;
        failover_strategy.retry_count = retries;
        failover_strategy.retry_delay = retry_delay;

 // Creating a session requires at least a URL. Creating a
 session
        // initiates a connection with Diffusion.
        SESSION_T *session;
        DIFFUSION_ERROR_T error;
        session = session_create(url,
                                 hash_get(options, "principal"),

 credentials_create_password(hash_get(options, "credentials")),
                                 &foo_listener,
 &failover_strategy, &error);
        if(session == NULL) {
                fprintf(stderr, "TEST: Failed to create session
\n");
                fprintf(stderr, "ERR : %s\n", error.message);
                return 1;
        }

 // Register handlers for callbacks we're interested in relating
 to
 // the fetch request. In particular, we want to know about the
 topic
 // messages that are returned, and the status message which
 tells
 // us when all messages have been received for the selector
 (or, if
 // something went wrong.)
        FETCH_PARAMS_T params = {
                .selector = topic,
                .on_topic_message = on_topic_message,
                .on_fetch = on_fetch,
                .on_status_message = on_fetch_status_message
        };

 // Issue the fetch request.
 fetch(session, params);

 // Wait for up to 5 seconds for the results to come in.
        sleep(1);

 // Clean up politely.
        session_close(session, &error);

        return 0;
}
```

Classic API

Diffusion provides a number of Application Programming Interfaces (APIs) which allow user-written applications to make use of Diffusion.

The APIs documented in this section are still supported for version 5.5, but in future is replaced by the new API. For more information, see Unified API.

Where an API is not available for a particular language a user can still communicate with Diffusion through a TCP socket based connection using the Diffusion Protocol.

Table of Classic APIs

Diffusion provides Classic APIs in a variety of languages. Each of these APIs is supported over a set of protocols and in a specific level of the language.

Note: We recommend that you use the Unified API where it is available for your platform.

	Publisher	Client	Implementation version	Protocols
Java	YES	YES	Java 7 or Java 8	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)
.NET		YES	.NET 4.5	• DPT, DPTS • WS, WSS • HTTP, HTTPS (Full duplex)
JavaScript		YES	JavaScript V1.3	• Native: WS, WSS, HTTP, HTTPS, HTTP Streaming • Flash: DPT, HTTP, HTTPS, HTTP Streaming • Silverlight: DPT, HTTP, HTTPS
Flash		YES	ActionScript V3.0 (Flex 3.0)	• DPT, DPTS • HTTP, HTTPS • HTTPC, HTTPCS
Silverlight		YES	Silverlight V4.0	• DPT • HTTP, HTTPS • HTTPC, HTTPCS

	Publisher	Client	Implementation version	Protocols
iOS		YES	iOS v5.1.1, device architectures: armv7, armv7s, arm64, simulator architectures: i386, x86_64	• DPT, DPTS • WS, WSS
Android		YES	Android 2.2	• DPT, DPTS

Tier 2 APIs supported on a best effort basis

	Publisher	Client	Implementation version	>Protocols
C		YES		• DPT

For more information on the transports supported by the clients, see Protocols supported on page 126

Table 62: Feature matrix

Feature	Java	.NET	JavaScript	C	Flex	Silverlight	Java ME	iOS	Android
Asynchronous connect		YES	YES		YES	YES		YES	
Simple connect	YES	YES	YES	YES	YES	YES	YES	YES	YES
Connect with topics (variable string args)	YES	YES	YES						
Connect with topicSet	YES	YES							
Connect with multiple server details	YES	YES		YES	YES	YES	YES	YES	YES
Connection cascading	YES	YES		YES	YES	YES	YES	YES	YES
Connection failover	YES	YES			YES	YES	YES	YES	YES
Connection load balancing	YES	YES		YES	YES	YES	YES	YES	YES
Reconnect	YES	YES	YES	YES	YES	YES	YES	YES	YES
Ping server	YES	YES	YES	YES	YES	YES	YES	YES	YES
Send topic load message	YES	YES	YES				YES		YES
Send delta message	YES	YES	YES	YES	YES	YES	YES	YES	YES
(Auto) message acknowledgment	YES	YES	YES	YES	YES	YES	YES	YES	YES

Feature	Java	.NET	JavaScript	C	Flex	Silverlight	Java ME	iOS	Android
(Manual) message acknowledgment	YES	YES	YES	YES	YES	YES	YES	YES	YES
Subscribe with topics (variable string args)	YES	YES					YES		YES
Subscribe with topicSet	YES	YES							
Fetch with topicSet	YES	YES							
Fetch	YES	YES	YES	YES	YES	YES	YES	YES	YES
Unsubscribe with topics (variable string args)	YES	YES							
Unsubscribe with topic	YES	YES	YES	YES	YES	YES	YES	YES	YES
Unsubscribe with topicSet	YES	YES							
Add topic listener	YES	YES	YES	YES	YES	YES	YES	YES	YES
Remove topic listener	YES	YES	YES	YES	YES	YES	YES	YES	YES
Create service topic handler	YES	YES	YES	YES	YES	YES	YES	YES	YES
Create paged topic handler	YES	YES	YES	YES	YES	YES	YES	YES	YES
Create topic notify topic handler	YES	YES	YES		YES		YES	YES	YES

Related Links

Diffusion APIs on page 48
Diffusion provides application programming interfaces (APIs) that you can use to create applications that interact with the Diffusion server.

Java API

The Java API comprises a number of packages all subordinate to the main `com.pushtechnology.diffusion.api` package.

Full API documentation is issued with the product. For information about how to use the API components, see the API documentation. The following table is a guide to which packages contain the major API components.

Table 63: Java APIs

API area	Relevant packages
Publishers	The main classes for writing a publisher are in the `publisher` package.

API area	Relevant packages
	General classes relating to topic handling are in `topic` Classes related to messages are in `message`. All classes relating to the use of TopicData are in the `data` package. Classes relating to the use of conflation are in the `conflation` package. For more information, see Publishers.
Client	The client API classes are in the `client` package and makes use of the `connection` package for connection details. Client applications also make use of `config` for configuration and `message` for message handling. The `threads` package also applies for thread pool handling.
Server	Classes relating to the Diffusion server (running it within an application) are located in the `server` package and the configuration of the server is done with the classes in the `config` package.
Web server	Classes relating to implementing Diffusion web server functionality are in the `webserver` package.
Configuration	The `config` package contains all configuration classes used for configuring the Diffusion server and/or within client APIs.
System management	System management classes are located in the `management` package.
Statistics	Classes related to statistics are in the `statistics` package.
WhoIs	If you want to implement your own WhoIs provider, the classes required are in the `whois` package.
General	General purpose classes relating to exceptions and logging as well as some useful utility classes are located directly under the `api` package itself.

Client API

The client API provides the ability to connect to a Diffusion server as an external client from within any Java application.

How to use the Java client API

There is a single class called `ExternalClientConnection` which can be instantiated with the required connection details and used to make an actual connection.

The connection class is of the generic type `ServerConnection` and as such, once a connection is made, any notifications or messages from the server are passed through the `ServerConnectionListener` interface.

The topic or topics to subscribe to can be specified when connecting or at any time after connection.

The `ServerConnectionListener` specified will receive all messages for all topics. However, any number of additional topic listeners can be specified and messages for different topics routed to different listeners as required.

The API permits the following types of connection to be specified by using the `ServerDetails` (see connection package) specified when configuring the connection object:

Table 64: Connection types

TCP	For a standard connection over TCP/IP. This must connect to a standard client connector.
SSL	For a secure TCP/IP connection over SSL. This must connect to a client connector with SSL enabled
HTTP	For a connection using HTTP
HTTP/SSL	For a connection using HTTP over SSL.

By specifying more than one `ServerDetails`, fallback connections can be specified. If the first connection does not succeed, the second is tried, and so on.

For a detailed description of the API see the issued API documentation (in docs directory).

Authorization credentials

If authorization credentials are required by the Diffusion server, these are set at the `ConnectionDetails` level and used for all `ServerDetails`. Credentials can be set in a `ServerDetails` by creating a `Credentials` object and using `setCredentials` before connecting.

Credentials can also be sent to the server after connection using the method `sendCredentials` in `ExternalClientConnection`. In this case the credentials can be rejected by the server, in which case this is notified on the `serverRejectedCredentials` method of each `ServerConnectionListener`.

Certificates

Diffusion Java clients use certificates to validate the security of their connection to the Diffusion server. The client validates the certificate sent by the Diffusion server against the set of certificates trusted by the .

If the certificate sent by the Diffusion server cannot be validated against any certificates in the set trusted by the , you must set up a trust store for the client and add the appropriate certificates to that trust store.

Diffusion is authenticated using the certificates provided by your certificate authority for the domain you host the Diffusion server on.

1. Obtain the appropriate intermediate certificate from the certificate authority.
2. Use keytool to create a trust store for your client that includes this certificate.

   For more information, see https://docs.oracle.com/cd/E19509-01/820-3503/ggfka/index.html
3. Use system properties to add the trust store to your client.

   For example:

```
System.setProperty("javax.net.ssl.trustStore",
 "truststore_name");
```

   Or at the command line:

```
-Djavax.net.ssl.keyStore=path_to_truststore
```

Reconnection

If a client unexpectedly loses connection, it can try to reconnect using the `reconnect` method. If the server has specified keep-alive for the connector, the client can pick up the same session as before and receive all messages that were queued for it whilst disconnected. The topic state (that is, which topics the client is subscribed to) is also re-established on reconnection. If unable to reconnect, a new connection is established with the same topic set as used on the original connection. Successful reconnection or connection is notified on the normal `serverConnected` method and you can determine which has occurred using the `ServerConnection.isReconnected()` method.

There is no guarantee that messages in transit at the time of the disconnection will be redelivered. However, all messages marked as requiring acknowledgment by the server are delivered.

Failover

The Java client supports autofailover. For more information, see Client failover on page 715.

Special features

Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Service topic data handling

Where service topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Example: Simple client class

The following example shows a simple client class which sends a message containing "Hello" to the server and logs all messages it receives, until it receives a message from the server (Publisher) asking it to stop. It tries to connect through TCP first but if that fails it tries HTTP.

```
public class ClientApplication implements
 ServerConnectionListener {

    private static final Logger LOG =
        LoggerFactory.getLogger(ClientApplication.class);

    private ExternalClientConnection theConnection;

    public ClientApplication() throws APIException {
        // Create Connection
        theConnection=
            new ExternalClientConnection(
                this,
                "ws://diffusion.example.com:80",
                "http://diffusion.example.com:80");
        // Connect, subscribing to a single topic
        theConnection.connect("MyTopic");
        // Send a message
        TopicMessage message =
 theConnection.createDeltaMessage("MyTopic");
        message.put("Hello");
        theConnection.send(message);
    }

    public void messageFromServer(
```

```
    ServerConnection serverConnection,
    TopicMessage message) {
        LOG.info("Message Received : {}",message);
        try {
            if (message.asString().equals("STOP")) {
                theConnection.close();
            }
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    public void serverConnected(ServerConnection
 serverConnection){
        LOG.info("Connected to Server :
 {}",serverConnection);
    }

    public void serverTopicStatusChanged(
    ServerConnection serverConnection,
    String topicName,
    TopicStatus status) {
        LOG.info(
            "Topic {} at {} status changed to {}",
            topicName,serverConnection,status);
    }

    public void serverRejectedCredentials(
    ServerConnection serverConnection,
    Credentials credentials) {
        LOG.info("Server Rejected Credentials :
 {}",serverConnection);

    }

    public void serverDisconnected(ServerConnection
 serverConnection) {
        LOG.info("Disconnected from Server :
 {}",serverConnection);
    }

}
```

Related Links

Using Maven to build Java Diffusion applications on page 786
Apache Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

.NET API

The .NET API comprises a number of packages.

Client API

The ExternalClient API provides the ability to connect to a Diffusion server as an external client from within any .NET application.

There is a single class called `ExternalClient` which can be instantiated with the required connection details and used to make an actual connection.

The topic or topics to subscribe to can be specified when connecting or at any time after connection.

When a connection object is instantiated, subscribe to the `InboundMessageReceived` delegate, which receives all messages for all topics.

The API permits the following types of connection to be specified by using the `ServerDetails` specified when configuring the connection object:

Table 65: Types of connection that can be specified from the .NET client

TCP	For a standard connection over DPT. This connects to the External Client Connector.
TCPSSL	For a secure TCP/IP connection over DPTS. This connects to the External Client Connector.
HTTP	For a connection using HTTP protocol
HTTPSSL	For a secure connection using HTTP protocol
WEBSOCKET	For a connection using WebSocket protocol
WEBSOCKETSSL	For a secure connection using WebSocket protocol

For a detailed description of the API, see the issued documentation (in docs directory).

The following example shows a simple client class which sends a message containing 'Hello' to the server until it receives a message from the server (Publisher) asking it to stop.

```
    public class ClientApplication : IServerConnectionListener,
 ITopicListener
 {
  #region Fields

  private readonly
 PushTechnology.DiffusionExternalClient.ExternalClient
 theClient;

  #endregion // Fields

  #region Constructor

  public ClientApplication()
  {
   var connectionDetails =
    ConnectionFactory.CreateConnectionDetails( "ws://
diffusion.example.com:80", "http://diffusion.example.com:80" );

   connectionDetails.Topics = new TopicSet("MyTopic");

   theClient = new
 PushTechnology.DiffusionExternalClient.ExternalClient(connectionDetails);
```

```
   // Add a topic listener – we are listening to all messages
 for this example, but individual topics can
   // also be used as selectors
   theClient.AddGlobalTopicListener( this );

   // Now connect – this is an asynchronous process, so we have
 to wait until ServerConnected is invoked
   theClient.Connect();
  }

  #endregion // Constructor

  #region Implementation of IServerConnectionListener

  /// <summary>
  /// Notification of connection.
  ///
  /// This is called when a connection to a server is
 established.
  /// </summary>
  /// <param name="connector">The server connector.</param>
  public void ServerConnected( IDiffusionClientConnector
 connector )
  {
   Console.WriteLine( "Connected to server: " + connector );

   // Send a message as we are now connected
   ITopicMessage message =
 theClient.CreateDeltaMessage( "MyTopic" );

   // Populate the message
   message.Put( "Hello" );

   // Send the message to the Diffusion server
   theClient.SendMessage( message );
  }

  /// <summary>
  /// Notification that the status for a topic that was
 subscribed to has changed.
  /// </summary>
  /// <param name="connector">The connector.</param>
  /// <param name="topicName">The name of the topic on which the
 status has changed.</param>
  /// <param name="statusType">The topic status change type.</
param>
  public void
 ServerTopicStatusChanged( IDiffusionClientConnector connector,
 string topicName, TopicStatusChangeType statusType )
  {
   Console.WriteLine(
    string.Format( "Topic status for '{0}' changed to '{1}'.",
 topicName, statusType ));
  }

  /// <summary>
  /// Notification of rejected credentials from the server.
  /// </summary>
  /// <param name="connector"></param>
  /// <param name="credentials"></param>
```

```
 public void
ServerRejectedCredentials( IDiffusionClientConnector connector,
V4Credentials credentials )
 {
  Console.WriteLine( "Server rejected credentials.");
 }

 /// <summary>
 /// Notification of disconnection.
 ///
 /// The reason for the disconnection can be established by
checking the state of the connection
 /// using IDiffusionClientConnector.State.
 /// </summary>
 /// <param name="connector">The server connector.</param>
 /// <param name="args">The arguments which can be interrogated
for the state and details of a server closure.</param>
 public void ServerDisconnected( IDiffusionClientConnector
connector, ServerClosedEventArgs args )
 {
  Console.WriteLine( "Disconnected from server.");
 }

 #endregion

 #region Implementation of ITopicListener

 /// <summary>
 /// Handles a message received from an IMessageSource.
 ///
 /// This handles an incoming message from a specified source.
 /// </summary>
 /// <param name="source">The message source.</param>
 /// <param name="message">The message.</param>
 public bool HandleTopicMessage( IMessageSource source,
ITopicMessage message )
 {
  if (message.AsString().Equals("STOP"))
  {
   theClient.Disconnect();
  }

  return false;
 }

 #endregion
}
```

Connection events

Events that are invoked when a connection to the Diffusion server is established, fails, or is lost are invoked synchronously.

The following connection events are invoked synchronously by the .NET client library:

- `DiffusionServerConnected` is invoked when the client library successfully establishes a connection to the Diffusion server
- `DiffusionServerConnectionFailed` is invoked when the client library is unable to establish a connection to the Diffusion server
- `DiffusionServerDisconnected` is invoked when an established connection to the Diffusion server is lost

Because these events are invoked synchronously, do not perform any long-running or blocking operations on these event threads. If you want to make another connection attempt after one of these events is invoked, create another thread to perform this task.

JavaScript API

The JavaScript API provides web developers with a simple means of connecting to and interacting with a Diffusion server from within a web browser. The API takes care to select the most appropriate underlying transport from those available.

For the list of supported web browsers, see Browsers supported on page 127.

Using the JavaScript Classic API

The JavaScript client library is located in the `clients/js` directory of the Diffusion installation. Two versions are provided, an uncompressed file, and a minimized file with the extension `min.js`.

To enable the Diffusion client for production use, host the `diffusion-js-classic.js` client library on a dedicated web server and load the client library into your web page:

```
<script type="text/javascript" src="library_location/diffusion-
js-classic.js"></script>
```

Dependent transport files

The JavaScript client depends on other files located in the same directory: `clients/js/diffusion-flash.swf` and `clients/js/diffusion-silvelight.xap`. These files provide Flash and Silverlight transport capabilities, respectively. Removing these files prevents the JavaScript client from using these transports.

You can configure the JavaScript client to point to specific versions of both the Flash and the Silverlight transports. This is available through the connection details.

```
var connectionDetails = new DiffusionClientConnectionDetails();

connectionDetails.libPath = "/lib/js/diffusion";
connectionDetails.libFlashPath = "diffusion-flash.swf";
connectionDetails.libSilverlightPath = "diffusion-
silverlight.xap";
```

Connection details

The `DiffusionClientConnectionDetails` object has over 20 attributes that change the way that the client behaves. Any attributes that are not set are provided with default values when used.

You can provide an anonymous object instead of instantiating a new `DiffusionClientConnectionDetails` object.

Connecting

A Diffusion client is a singleton with global scope. It can be called from anywhere. To connect to Diffusion, call the connect method. The method takes two parameters, first is the connection details and the optional second object is the client credentials. The following example is using an anonymous connection object:

```
DiffusionClient.connect({
   debug : true,
   onCallbackFunction : function(isConnected) {
```

```
        console.log("Diffusion client is connected: " +
 isConnected);
   }
})
```

If the client connection fails, the JavaScript client attempts to connect through other protocols. This is called protocol cascading.

Credentials

Credentials can either be supplied on the connect method or set separately using the `DiffusionClient.setCredentials(...)` method. These credentials are used for all transports that are used to connect to Diffusion. The `DiffusionClientCredentials` object is a simple one of username and password attributes.

```
// Connect with supplied credentials
DiffusionClient.connect({...}, {
 username : "foo",
 password : "bar"
});
```

```
// Connect, and send credentials later
DiffusionClient.connect({
 onCallbackFunction : function() {
  DiffusionClient.sendCredentials({
   username : "foo",
   password : "bar"
  });
 }
});
```

If authentication of the client connection fails, the JavaScript client attempts to protocol cascade and to connect through a different protocol with the same credentials. Use the `onConnectionRejectFunction` to close the client connection and prevent this from happening.

Events

The connection details have attributes that are listeners for certain events. If these are set in the connection object, they are called when these events happen.

Table 66: JavaScript functions called on events

Function	Description
`onDataFunction`	This function is responsible for handling messages from Diffusion. This function is called with an argument of `WebClientMessage`. This function is called even if there is a topic listener in place for a topic.
`onBeforeUnloadFunction`	This function is called when the user closes the browser or navigates away from the page.
`onCallbackFunction`	This function is called when Diffusion has connected, or exhausted all transports and cannot connect. This function is called with a boolean argument.

Function	Description
`onInvalidClientFunction`	This function is called when an invalid Diffusion operation is called, for instance if `Diffusion.subscribe` is called before `Diffusion.connect`
`onCascadeFunction`	This function is called when the client cascades transports. The function is called with an argument of the {String} transport name or NONE if all transport are exhausted.
`onPingFunction`	This function is called with an argument of `PingMessage` when the ping response has been returned from the server.
`onAbortFunction`	This function is called when the Diffusion server terminates the client connection (or the connection has been banned).
`onLostConnectionFunction`	This function is called when the client loses connection with the Diffusion server
`onConnectionRejectFunction`	This function is called when the client connection is rejected by the Diffusion server because of incorrect credentials. Use this function to close the connection when the authentication fails and prevent the client attempting to connect over a different protocol with the same incorrect credentials.
`onMessageNotAcknowledgedFunction`	This function is called when a message that is requested as Acknowledge did not respond in time.
`onServerRejectedCredentialsFunction`	This function is called after a `DiffusionClient.sendCredentials` and the server rejected the credentials.
`onTopicStatusFunction`	This function is called if the status of a subscribed topic changes.

Receiving messages

The `onDataFunction` with a class called `WebClientMessage` contains only one message even if the messages sent from the Diffusion server are batched. If this is the case, this method is repeatedly called until all of the messages are exhausted. The `WebClientMessage` class wraps the message sent from the Diffusion server with utility methods like `isInitialTopicLoad()` and `getTopic`. See the jsdoc for the full list of utility methods.

Sending messages

There are two ways of sending messages to the Diffusion server:

- The `DiffusionClient.send(topic, message)` method
- The `sendTopicMessage` method

If user headers are required, it is best to use the `TopicMessage` class. The following example shows how to send a message using the `TopicMessage` class.

```
var topicMessage = new TopicMessage("Echo", "This is a
 message");
topicMessage.addUserHeader("Header1");
topicMessage.addUserHeader("Header2");

DiffusionClient.sendTopicMessage(topicMessage);
```

Subscribing and unsubscribing

To subscribe and unsubscribe use the `DiffusionClient.subscribe(topic)` and `DiffusionClient.unsubscribe(topic)` methods respectively. The parameter can be a topic, a topic selector, or a comma-delimited list of topics

Topic listeners

During the lifetime of the connection, it might be required to have modular components that are notified about topic messages. These are topic listeners. A topic listener calls a supplied function with a `WebClientMessage` object when the topic of the message matches the pattern supplied. It is also worth noting that the `onDataMessage` function is called as well as the topic listener function. You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, you can use the following example code:

```
DiffusionClient.addTopicListener(""^Logs$", onDataTradeEvent);
```

Note:

The characters `^` `$` are regular expression characters. For more information, see Regular expression. The preceding code example means that the listener is only interest in receiving the message event if the topic is Logs. If the following example code is used, any topic name that has Logs in it matches.

```
var listenerRef = DiffusionClient.addTopicListener("Logs",
 onLogEvent, this);
```

Retain the listener reference if you want to remove the topic listener at a later date.

Failover

The JavaScript client does not support autofailover. You can still implement this using the `onLostConnectionFunction`.

Special Features
Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Reconnecting with the JavaScript API

The JavaScript API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

The JavaScript API listens for pings from the server and raises an event if the connection to the server is lost.

To enable the liveness monitor, set the `enableLivenessMonitor` parameter to true inside the client connections details. For example:

```
var connectionDetails = { ...
    enableLivenessMonitor : true,
... };

DiffusionClient.connect(connectionDetails);
```

The Diffusion server sends out pings at regular intervals. The length of this interval is configured at the server by using the `system-ping-frequency` element in the `Connectors.xml` configuration file.

The liveness monitor in the ActionScript client library listens for the pings from the server and uses them to estimate the ping interval. The liveness monitor takes an average of the time between the pings it receives to estimate the ping interval. It revises this estimation each time it receives a ping, until it has received ten pings. After ten pings the liveness monitor has obtained the estimated ping interval that it uses for the rest of the client session.

If the liveness monitor does not receive a ping within a time interval equal to twice the length of the estimated ping interval, it considers the connection lost and raises a connection lost event.

Connection lost events can be raised by the liveness monitor or triggered by other events, such as an unexpectedly closed connection.

You can implement an event listener in your client that listens for a connection lost event and reacts to it by using the `reconnect()` method to reestablish the connection.

Reconnection example

To reconnect after you lose connection, you must use the `reconnect()` method. You cannot reconnect an aborted client.

The following code shows how to setup an event listener for connection events, if the connection has been lost how to reconnect and how to tell if you have successfully reconnected the client.

```
var connectionDetails = {
        onCallbackFunction : function( isConnected,
 isReconnect) {
                if( !isConnected && isReconnect ) {
                        DiffusionClient.reconnect();
                }
        },
        onLostConnectionFunction : function() {
                DiffusionClient.reconnect();
        }
};

DiffusionClient.connect(connectionDetails);
```

Related Links

Client reconnection on page 711
You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep alive the client session.

http://docs.pushtechnology.com/docs/5.5.0/js-classic/index.html

Service topic data in JavaScript

The JavaScript API provides a basic interface for using service topics.

The API consists of a service topic handler to process responses and using the generic `DiffusionClient.command(...)` method to send service requests.

The common sequence to follow is:

1. Add a topic listener, to capture the service topic load message
2. Subscribe to the service topic
3. With the ITL from the service topic create a service topic handler
4. Remove the topic listener
5. Send command messages to the service
6. Process any response in the function passed to the handler

To create a handler using the `DiffusionClient.createServiceTopicHandler(TopicMessage, function)` you must pass in the ITL of the service topic and the function that is called when a service response is received. This function will be called with a `CommandMessage` as an argument.

To make service requests you must use the `DiffusionClient.command(string,string,TopicMessage)` method to send command messages. The first string is the command to send. The second string is a correlation ID for the response. The TopicMessage is the message sent to the client with the correct topic and any additional headers or payload you want to send in the request.

Use an ordinary topic listener to get the ITL to create the service topic handler. This listener is not required for any subsequent message processing and you are encouraged to remove it after you have the ITL.

You must generate a unique value for the correlation ID.

Paged topic data in JavaScript

The JavaScript API provides an interface for using paged topics.

The API contains the following classes:

`PagedTopicHandler`

Provides methods that enable you to change and navigate the page view.

`PagedTopicListener`

Provides callbacks for when an action is performed on a page or the topic.

`PageStatus`

Contains values that describe the status of a page or topic.

For more information, see JavaScript Classic API documentation.

Handler methods

The following example code shows some of the handler methods wrapped in functions you can use to add buttons to a client's user interface.

```
// Get the next page in the topic
function next() {
 handler.next();
}

// Get the previous page in the topic
function prior() {
 handler.prior();
}
```

```
// Get the first page in the topic
function first() {
 handler.first();
}

// Get the last page in the topic
function last() {
 handler.last();
}

// Close the paged view of the topic
function pagedclose() {
 handler.close();
}
```

Listener methods

The following example code creates a `PagedTopicHandler` and implements the listener methods `add`, `page`, `ready`, `statusChanged`, and `update`.

```
var connectionDetails = {
 onDataFunction : function() {
 },
 onCallbackFunction : function() {
  // Creates the handler for a topic
  DiffusionClient.createPagedTopicHandler(topic, {
   ready : function(handler) {
    // Add here the code you want to run when the handler
    // is created.

    // For example, open a view that is 20 lines long and
    // contains the first page of data:
    handler.open(20, 1);
   },
   page : function(handler, status, lines) {
    // Add here the code you want to run when the page is
    // loaded.

   },
   update : function(handler, status, index, line) {
    // Add here the code you want to run when a line on
    // the current page is updated.

    // For example, refresh the page.
    handler.refresh();
   },
   statusChanged : function(handler, status) {
    // Add here the code you want to run when the status
    // of the current page changes.

    // For example, check whether the page is dirty and
    // refresh the page if this is true.
    if (status.isDirty){
     handler.refresh();
    }
   },
   add : function(handler, status, lines) {
    // Add here the code you want to run when a line is
    // added to the current page.
   }
```

```
  });
 },
 debug : true
};
```

Related Links

Paged record topics on page 58
A topic that maintains server-side state as a number of lines of record data. The schema defines the record metadata that defines the lines. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

Paged string topics on page 57
A topic that maintains server-side state as a number of lines of string data. A client can view the data as pages made up of one or more lines and can page forward and backward through the data.

ActionScript API

The ActionScript API is bundled in a library called `clients/flex/diffusion-flex.swc`.

This can be embedded into a Flex/Flash or Air application. Full asdoc is issued with the product so the sections below provide a brief outline of the uses for the classes and examples of their use. The ActionScript library is based on the event model. There are many different types of events that a DiffusionClient dispatches. These must be registered before notification happens.

Diffusion also provides a debug-friendly version of the library: `diffusion-flex-debug-`*`version`*`.swc`, where *version* is the Diffusion version number, for example 5.5.3. This version is larger, but you can embed it into you application to receive additional information from any stack traces.

Using the ActionScript API

DiffusionClient (`com.pushtechnology.diffusion.DiffusionClient`) is the main class that is used. This class enables the user to set all of the connection and topic information.

DiffusionClient

Connection example

```
import com.pushtechnology.diffusion.ServerDetails;
import com.pushtechnology.diffusion.ConnectionDetails;
import com.pushtechnology.diffusion.DiffusionClient;
import
 com.pushtechnology.diffusion.events.DiffusionConnectionEvent;
import
 com.pushtechnology.diffusion.events.DiffusionTraceEvent;
import
 com.pushtechnology.diffusion.events.DiffusionMessageEvent;
import
 com.pushtechnology.diffusion.events.DiffusionExceptionEvent;
import
 com.pushtechnology.diffusion.events.DiffusionPingEvent;

// Get a new DiffusionClient
var theClient:DiffusionClient = new DiffusionClient();

// Set everything to enable the cascading
```

```
var serverDetails:ServerDetails = new ServerDetails("https://
diffusion.example.com:443");
var connectionDetails:ConnectionDetails = new
 ConnectionDetails(serverDetails, "Trade");
connectionDetails.setCascade(true);

// Add the listeners
theClient.addEventListener(DiffusionConnectionEvent.CONNECTION,
 onConnection);
theClient.addEventListener(DiffusionMessageEvent.MESSAGE,
 onMessages);
theClient.addEventListener(DiffusionTraceEvent.TRACE,
 onTrace);
theClient.addEventListener(DiffusionExceptionEvent.EXCEPTION,
 onException);
theClient.addEventListener(DiffusionPingEvent.PING, onPing);

// Connect
theClient.connect(connectionDetails);
```

Setting credentials

If credentials are required by the Diffusion server then use the `setCredentials` method on the `DiffusionClient` class. The `DiffusionClientCredentials` class takes a constructor argument of username and password. Please bear in mind, that these are only tokens and can contain any information that the `AuthorisationHandler` requires.

```
var credentials:DiffusionClientCredentials = new
 DiffusionClientCredentials(username, password);
theClient.setCredentials(credentials);
```

Connection event

The connection event contains information about the success of the connection attempt. Below is a coding example of the possibilities for the connect event.

```
public function onConnection(event:DiffusionConnectionEvent) :
 void {
 if (event.wasConnectionRejected()) {
  theClientIDBox.text = "Connection Rejected by Diffusion
 Server";
 } else if (event.wasClientAborted()) {
  theClientIDBox.text = "Connection aborted";
 } else if (event.isConnected()) {
  theClientIDBox.text = event.getClientID();
  theConnectedTransportLabel.text =
 theClient.getTransportMode();
 } else {
  theClientIDBox.text = "Connection failed " +
 event.getErrorMessage();
 }
}
```

You can receive a connection event after you have successfully connected, which might be because of a lost connection, or in the case of client aborted the Diffusion server has deliberately closed the client connection. This normally means that a publisher has aborted the connection and the client must not try and connect again.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `DiffusionMessageEvent` is dispatched. Contained in the event is a TopicMessage object `TopicMessage` (com.pushtechnology.diffusion.TopicMessage). This class contains helper methods that surround the message itself, like getTopic() and isInitialTopicLoad. For more information, see the API documentation.

```
public function onMessages(event:DiffusionMessageEvent) : void {
 var message:TopicMessage = event.getTopicMessage();
 ...
```

Subscriptions

Once the client has connected, you can issue subscribe and unsubscribe commands. The subscribe and unsubscribe methods take a string format, that can be a topic selection pattern, a list of topics that are comma delimited or a single topic.

Send

Once connected a client can send messages to the Diffusion server on a particular topic. To do this, use the send method.

```
theClient.send("Fred","Hello publisher that looks after Fred");
```

In the example above, the publisher that looks after topic Fred receives a `messageFromClient` notification. If a message with user headers or encoding is required, you must use the `sendTopicMessage` method. A TopicMessage (`com.pushtechnology.diffusion.TopicMessage`) allows for the setting of user headers and message encoding

Ping

The client can ping the Diffusion server. To receive the Ping response, the listener is added to the client.

```
theClient.addEventListener(DiffusionPingEvent.PING, onPing);
```

The resulting ping event has two attributes in it, firstly the time taken to do a round trip from the client to the Diffusion server and back again. The second attribute is how many items are currently in the client queue at the server. This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the life time of the connection, it might be required to have modular components notified about topic messages – these are topic listeners. A topic listener calls a supplied function with a TopicMessage object when the topic of the message matches the pattern supplied. The topic listeners are called in the order that they are added, and before the default DiffusionMessageEvent.MESSAGE, that is called as well as the topic listener event. You can have many topic listeners on the same topic pattern if required. The function supplied in charge of

processing the message can signal that a message is consumed, returning TRUE. In this case, this message is not relayed to subsequent TopicListeners and the default listener. For example, if you want to be notified about a particular topic, use the following code:

```
var listenerRef:String = theClient.addTopicListener("^Logs$",
 theLogsDataGrid.onMessage);
```

Note the syntax here, the ^ $ are regex pattern strings, the above means that the listener is only interested in receive the message event if the topic is Logs. If the following was issued.

```
var listenerRef:String = theClient.addTopicListener("Logs",
 theLogsDataGrid.onMessage);
```

Any topic name that has “Logs” in it matches. You must store the reference to remove the topic listener at a later date.

Timed topic listeners

A timed topic listener calls a supplied function with an array of topicMessage objects when the topic of the message matches the pattern, and only if the time supplied by the arguments has expired. Otherwise, the TopicMessage is stored until the time expired.

Note:

The function in charge of processing the message cannot determine if a message is consumed as you can do in a topic listener. For example, if you want to be notified about a particular topic, use the following code:

```
var timedListenerRef:String =
 theClient.addTimedTopicListener("^Logs$",
 theLogsDataGrid.onMessage, 2000, false);
```

The third parameter is the frequency at which the function supplied is called. The optional fourth parameter can be set if this function must be called, even if no messages are stored.

Failover

The ActionScript client supports autofailover. For more information, see ActionScript failover documentation.

Special features

Paged topic data handling

Where paged topic data is in use at the server there are features within the client API which simplify the handling of messages to and from such a topic.

Reconnecting with the ActionScript API

The ActionScript API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

Liveness monitor

The ActionScript API implements a liveness monitor that listens for pings from the server and raises an event if the connection to the server is lost.

Before you make a connection to the Diffusion server, enable the liveness monitor by using the `enableLivenessMonitor()` method. For example:

```
client.enableLivenessMonitor(true);
client.connect(connectionDetails);
```

The Diffusion server sends out pings at regular intervals. The length of this interval is configured at the server by using the `system-ping-frequency` element in the `Connectors.xml` configuration file.

The liveness monitor in the ActionScript client library listens for the pings from the server and uses them to estimate the ping interval. The liveness monitor takes an average of the time between the pings it receives to estimate the ping interval. It revises this estimation each time it receives a ping, until it has received ten pings. After ten pings the liveness monitor has obtained the estimated ping interval that it uses for the rest of the client session.

If the liveness monitor does not receive a ping within a time interval equal to twice the length of the estimated ping interval, it considers the connection lost and raises a `DiffusionConnectionEvent` whose `hasLostConnection()` method returns true.

You can implement an event listener in your client that listens for this event and reacts to it by using the `reconnect()` method to reestablish the connection.

Warning:

The liveness monitor relies on server pings being received at regular intervals. If the server pings the client in addition to the regular pings, these additional pings can cause the liveness monitor to make an incorrect estimate of the ping interval. Because this incorrect estimate is shorter than the correct ping interval, this can cause the liveness monitor to incorrectly consider a connection lost.

To avoid this problem, if you are using the liveness monitor, ensure that you do not ping the client from a publisher or from the Introspector.

Reconnection example

To reconnect, you must use the `reconnect` method when you lose a connection. You cannot reconnect an aborted client.

The following code shows how to setup an event listener for connection events, if the connection has been lost how to reconnect and how to tell if you have successfully reconnected the client.

```
var client:DiffusionClient = createClient();

function onConnectionEvent(event:DiffusionConnectionEvent) {
 if (event.hasLostConnection()) {
  client.reconnect();
 }
 else if (event.isConnected()) {
  if (event.isReconnected()) {
   // Successful reconnection
  }
 }
}

function createClient():DiffusionClient {
 var client:DiffusionClient = new DiffusionClient();
 client.addEventListener(DiffusionConnectionEvent.CONNECTION,
 onConnectionEvent);
 return client;
```

```
}
```

Related Links

Client reconnection on page 711
You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep alive the client session.

http://docs.pushtechnology.com/docs/5.2.0/flex/index.html

Silverlight API

The Silverlight API is bundled in an assembly called `clients/silverlight/PushTechnology.Transports.dll`. This can be embedded into a Silverlight application.

Full API documentation is issued with the product, so the sections below provide a brief outline of the uses for the classes and examples of their use. The Silverlight library is based on an asynchronous event model. There are a few events that the client object invokes. The client must subscribe to these events before notification can happen.

Using the Silverlight API

The `DiffusionClient` class is the main class that is used. This class enables the user to set all of the connection and topic information.

Instantiation and connection example

```
 theClient = new DiffusionClient( Dispatcher );

   // Instantiate the server details object and the initial
 topic to subscribe to
   ServerDetails details = new ServerDetails( "http://
localhost:8080", "SpotOnly" );

   // Add the server details to the client
   theClient.AddServerDetails( details );

            // Add the event listeners
            theClient.ConnectionStatus +=
 DiffusionConnectionStatus;
            theClient.MessageReceived +=
 theClient_MessageReceived;

            // Now connect
            theClient.Connect();
```

Setting credentials

If credentials are required by the Diffusion server then use the `Credentials` property on the `DiffusionClient` class. The `DiffusionClientCredentials` class takes a constructor argument of `userName` and `password`. Please bear in mind that these are only tokens and can contain any information that the `AuthorisationHandler` requires.

```
theClient.Credentials = new DiffusionClientCredentials(
    "username", "password" );
```

Rejection of credentials event

If the credentials are rejected by the Diffusion server, a `ServerRejectedCredentials` event is fired. This can be subscribed to by using the following code:

```
theClient.ServerRejectedCredentials +=
   ServerRejectedCredentials;
```

Message not acknowledged event

When a message is created with the "acknowledge" flag, this event is fired when a message is not acknowledged by the Diffusion server within the specified time period. This can be subscribed to by using the following code:

```
theClient.MessageNotAcknowledged += MessageNotAcknowledged;
```

The ConnectionStatus event

The `ConnectionStatus` event contains information about whether the connection was successful. Here follows an example of the usage of this event.

```
 /// <summary>
       /// Called when the connection state to Diffusion
changes.
       /// </summary>
       /// <param name="sender"></param>
       /// <param name="e"></param>
       void DiffusionConnectionStatus(
           object sender,
           DiffusionConnectionStatusEventArgs e )
       {
           switch( e.StatusType )
           {
               case
DiffusionConnectionStatusType.ConnectionFailed:
               {
                   Dispatcher.BeginInvoke( () =>
                       MessageBox.Show( "Unable to connect to
Diffusion.  Diffusion reports: " + e.ExtraData,
                           "Connection failed",
MessageBoxButton.OK ) );
               }
               break;

               case
DiffusionConnectionStatusType.ConnectionReset:
               {
                   Dispatcher.BeginInvoke( () =>
                       MessageBox.Show( "The connection to
Diffusion has been reset.  Diffusion reports: " + e.ExtraData,
                           "Connection failed",
MessageBoxButton.OK ) );
               }
               break;
           }
       }
```

Note: You can receive a `ConnectionStatus` event after you have successfully connected; it might be because of a lost connection, or in the case of `ConnectionAborted`, the Diffusion server has closed the client connection. This normally means that a publisher has aborted the connection and the client must not try to connect again.

The MessageReceived event

When messages arrive from the Diffusion server on a subscribed topic, the `MessageReceived` event is fired. This event contains a sender and a `TopicMessageEventArgs` object which itself contains a `TopicMessage` object which can be interrogated to discover the contents of the received message.

The TopicStatusMessageReceived event

When the status of a topic changes on the Diffusion server, the `TopicStatusMessageReceived` event is fired. This event contains a sender and a `TopicStatusMessageEventArgs` object which contains the alias of the topic on which the status has changed. Currently, only the notification of the removal of a topic is implemented.

Subscriptions

After the client has connected, you can issue `Subscribe` and `Unsubscribe` commands. These commands take string arguments which can be a topic selection pattern, a list of topics that are comma-delimited, or a single topic.

Sending non-encoded topic messages

Once the client has connected, it can send messages to the Diffusion server on a particular topic. To do this, use either the `Send` or `SendTopicMessage` methods on the DiffusionClient object, as shown in the following code:

```
  theClient.Send( "Fred", "Hello, publisher that looks after
Fred" );

             TopicMessage message =
                 new TopicMessage( "Fred", "Hello, publisher
that looks after Fred" );

             theClient.SendTopicMessage( message );
```

Note: The `TopicMessage` itself contains methods to set (for instance) user headers and encoding, or the convenience methods described below can handle the alternate encoding scenarios.

In the above examples, the publisher that looks after the topic `Fred` receives a `messageFromClient` notification internally.

Sending an encrypted topic message

Sending an encrypted topic message is achieved by calling the `SendTopicMessageEncrypted` method on the `DiffusionClient` object by using the following code:

```
 theClient.SendTopicMessageEncrypted( new TopicMessage( "Fred",
"Hello, publisher that looks after Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be encrypted immediately prior to sending to the Diffusion server.

> **Note:** Because of the limitations of HTTP-based transports, attempting to send a message of this type results in a non-encoded message being sent.

Sending a compressed topic message

Sending a compressed topic message is achieved by calling the `SendTopicMessageCompressed` method on the `DiffusionClient` object by using the following code:

```
 theClient.SendTopicMessageCompressed( new
TopicMessage( "Fred", "Hello, publisher that looks after
Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be compressed immediately prior to sending to the Diffusion server.

> **Note:** Because of the limitations of HTTP-based transports, attempting to send a message of this type results in a non-encoded message being sent.

Sending a Base64-encoded topic message

Sending a Base64-encoded topic message is achieved by calling the `SendTopicMessageBase64` method on the `DiffusionClient` object to using the following code:

```
theClient.SendTopicMessageBase64( new TopicMessage( "Fred",
 "Hello, publisher that looks after Fred" ) );
```

This sets the relevant encoding flags on the message itself, and the message will be Base64-encoded immediately prior to sending to the Diffusion server.

Ping

A client can ping the Diffusion server. To process the ping response, the user monitors the `MessageReceived` event and checks for a message type of `PingServer`, as shown in the following code:

```
   private void HandleServerPingMessage( TopicMessageEventArgs
e )
 {
  var message = e.Message as PingMessage;

  if( message != null )
  {
   tbElapsedTime.Text = message.ElapsedTime.ToString();
   tbQueueSize.Text = message.QueueSize.ToString();
  }
 }
```

Fetch

Using the `fetch` method, a client can send a request to the Diffusion server for the current state of a topic, which returns a state message to the client. A client can do this even if not subscribed to the topic.

Topic listeners

During the lifetime of the connection, it might be required to have modular components that get notified about topic messages – these are known as topic listeners. A topic listener calls a supplied function with a `TopicMessage` object when the topic of the message matches the pattern supplied. It is also worth noting that the `OnMessageReceived` event is called as well as the topic listener event itself.

You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, use the following code:

```
instrumentListener = theClient.AddTopicListener(
            "^SpotOnly$", ProcessInstruments, this );
```

Note: The "^" and "$" characters are regular expression pattern strings; the above means that the listener is only interested in receiving the message if the topic is SpotOnly.

Enabling JavaScript method invoking

To call JavaScript functions (and they are permitted to do so by the Silverlight runtime), use the following method call:

```
theClient.InitialiseJavaScriptMethodInvoking( HtmlPage.Window );
```

Listening to internal transport debug messages

To subscribe to the internal log tracings of the Silverlight API, the user can subscribe to the `DiffusionTraceEvent` on the `DiffusionClient` object, as shown in the following code:

```
theClient.DiffusionTraceEvent += theClient_DiffusionTraceEvent;
```

This enables the user to monitor all internal debug messages within the Silverlight API.

iOS Classic API

The static libraries and header files that comprise the iOS API are provided in the file `diffusion-ios-`*`version`*`.zip`, where *version* is the version number, for example 5.5.3.

The iOS library is provided with the Diffusion server installation in the `clients/ios-classic` folder.

The iOS library uses the delegate model. There are a number of Diffusion events dispatched by a `DFClient` instance as Objective-C messages. To receive these events, provide an implementation conforming to the `DFClientDelegate` protocol.

The API documentation is also available as an Xcode docset. Once installed into Xcode the iOS client can be browsed within the **Xcode** Documentation viewer.

Support

Table 67: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
iOS Classic API	v5.1.1	• DPT, DPTS

Platform	Supported version	Supported transport protocols
	device architectures: armv7, armv7s, arm64 simulator architectures: i386, x86_64	• WS, WSS

Features are sets of capabilities provided by the API.

Table 68: Capabilities provided by the API

Feature set	Supported capabilities
Messaging	Send messages through a topic path to the message handler of that topic path Receive messages sent through a topic path
Pings	Ping the Diffusion server
Security	Change the security username and password associated with the client session
Topics	Subscribe to topics Receive updates on subscribed topics Fetch the current state of a topic Unsubscribe from topics

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 69: Capabilities provided by the library

Library capability	Supported
Connect with multiple server details	✓
Transport cascading	✓
Connection failover	✓
Connect through load balancer	✓
Reconnect	✓

Getting started with iOS

Create a client application within minutes that connects to the Diffusion server.

Before you begin

Ensure that the iOS client libraries are available on your development system. The libraries are included in the Diffusion installation, which is available from the following location: Get the iOS libraries from the Diffusion installation. Install Diffusion and get the `diffusion-ios-version.zip` file from the `clients/ios` folder of the installation. For more information, see Graphical installation.

About this task

These instructions have been created using Xcode 6.0.1.

Procedure

1. Extract the contents of the `diffusion-ios-version.zip` file to your preferred location for third-party SDKs for use within Xcode.

   For example, you might have a directory within your `Documents` folder for `code` within which you have a sub-directory for software development kits (`SDKs`). In this case, locate the iOS SDK for Diffusion in the following directory: `~/Documents/code/SDKs/diffusion-ios-version/`, where *version* is the version number, for example 5.5.3
2. Create an Xcode project for your Diffusion client.
   a) From the **File** menu, select **New** > **Project...**

      Xcode prompts you to **Choose a template for your new project**.
   b) Select **iOS** > **Application** on the left.
   c) Select **Single View Application** on the right and click **Next**.

      Xcode prompts you to **Choose options for your new project**.
   d) Configure your project appropriately for your requirements.

      The Diffusion iOS SDK does not work with Swift without additional code, so select `Objective-C` as the **Language**.

      For example, use the following values:

      - **Product Name**: `TestClient`
      - **Language**: `Objective-C`
      - **Devices**: `Universal`

   e) Click the **Next** button.

      Xcode prompts you to select a destination folder for your new project.
   f) Select a target folder. For example, `~/Documents/code/`, and click **Create**.
3. Import the Diffusion iOS SDK.

   Use the Xcode **Build Settings** to define the location of your Diffusion iOS SDK.
   a) Go to the **Build Settings** tab for the Project or Target.
   b) Click the plus sign (+) and select **Add User-Defined Setting**.
   c) Set the name of the user-defined setting to `DIFFUSION_ROOT` and the value to the top-level directory of your extracted Diffusion iOS SDK.

      We recommend that you use the Xcode `SRCROOT` property in order to provide a relative location. For example, `$(SRCROOT)/../SDKs/diffusion-ios-version` defines the location of the Diffusion iOS SDK as the directory given in step 1 on page 525
   d) Go to the **User Header Search Paths** (USER_HEADER_SEARCH_PATHS) setting and add the following value: `$(DIFFUSION_ROOT)/headers`

      Use the default, non-recursive option.

e) Go to the **Library Search Paths** (LIBRARY_SEARCH_PATHS) setting and add the following values:

- Debug configuration: `$(DIFFUSION_ROOT)/Debug-universal`
- Release configuration: `$(DIFFUSION_ROOT)/Release-universal`

Use the default, non-recursive option.

f) Go to the **Other Linker Flags** (OTHER_LDFLAGS) setting and add the following value:

```
-lDiffusionTransport
```

For more information, see https://developer.apple.com/library/mac/documentation/DeveloperTools/Reference/XcodeBuildSettingRef/1-Build_Setting_Reference/build_setting_ref.html.

4. Add the required system libraries

The Diffusion iOS SDK depends on Zlib and on ICU: International Components for Unicode. These libraries are not included with the linker requirements by default for new Xcode projects so you need to add them.

a) Go to **Target** > **Build Phases** > **Link Binary With Libraries**
b) Add `libz.dylib`
c) Add `libicucore.dylb`

The following libraries are included by default for new Xcode projects and are required by the Diffusion iOS SDK:

- `CFNetwork.framework`
- `Foundation.framework`
- `Security.framework`

5. Create a client that connects to the Diffusion server when the view controller loads. (`ViewController.m`)

```
#import "ViewController.h"
#import "diffusion.h"

@interface ViewController (DFClientDelegate)
 <DFClientDelegate>
@end

@implementation ViewController
{
    DFClient *_diffusionClient;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    NSURL *const serverURL = [NSURL URLWithString:@"ws://
diffusion.example.com:80"];
    DFServerDetails *const serverDetails = [[DFServerDetails
 alloc] initWithURL:serverURL error:nil];
    DFConnectionDetails *const connectionDetails =
 [[DFConnectionDetails alloc] initWithServer:serverDetails

                            topics:@"Assets/"

                  andCredentials:nil];
    _diffusionClient = [[DFClient alloc] init];
    _diffusionClient.delegate = self;
```

```
    [_diffusionClient setConnectionDetails:connectionDetails];
    [_diffusionClient connect];
}

@end

@implementation ViewController (DFClientDelegate)

-(void)onConnection:(const BOOL)isConnected
{
    NSLog(@"Diffusion %@connected.", (isConnected ? @"" :
 @"NOT "));
}

-(void)onMessage:(DFTopicMessage *const)message
{
    NSLog(@"Diffusion message: \"%@\" =\"%@\"", message.topic,
 message.records[0]);
}

// Implement other required methods
-(void)onAbort { }
-(void)onConnectionSequenceExhausted:(DFClient *const)client
 { }
-(void)onLostConnection { }
-(void)onMessageNotAcknowledged:(DFTopicMessage *const)message
 { }
-(void)onPing:(DFPingMessage *const)message { }
-(void)onServerRejectedConnection { }

@end
```

a) Import the `diffusion.h` header file.

   This file pulls in the other required header files.

b) Conform to the `DFClientDelegate` protocol, using a category with the same name to enhance readability.

c) In the `viewDidLoad` method, assign `serverURL` to point to the Diffusion server using the DPT protocol.

d) Create a `DFServerDetails` object, `serverDetails`. Use the `initWithURL` method to wrap `serverURL`.

   Change the URL from that provided in the example to the URL of the Diffusion server.

e) Create a `DFConnectionDetails` object, `connectionDetails`. Use the `initWithServer` method to include `serverDetails`. Request a default, recursive subscription to the Assets topic.

f) Define a `_diffusionClient` instance variable.

g) Assign `self` to the `delegate` property of `_diffusionClient`.

h) Use the `setConnectionDetails` method to include `connectionDetails`.

i) Use the `connect` method to connect `_diffusionClient` to the Diffusion server.

j) Implement the `DFClientDelegate` category.

k) In the `DFClientDelegate` implementation, implement `onConnection:` to perform the required actions when the client connects. For example, log that the connection was successful.

l) In the `DFClientDelegate` implementation, implement `onMessage:` to perform the required actions when a message is received. For example, log the message content.

m) In the `DFClientDelegate` implementation, implement the other required methods.

   - `onAbort:`

- `onConnectionSequenceExhausted:`
- `onLostConnection:`
- `onMessageNotAcknowledged:`
- `onPing:`
- `onServerRejectedConnection:`

These implementations can be empty.

Results

The client connects to the Diffusion server. It receives a callback from the Diffusion iOS SDK though the `onConnection:` implementation. When connected the client receives topic messages (both for initial topic load and deltas) from the Diffusion iOS SDK through the `onMessage:` implementation.

Using the iOS Classic API

There are features, issues, and considerations that are specific to clients that are implemented using the iOS Classic API.

Diffusion Delegate

The Diffusion `Delegate` class is a custom class that must adhere to the `DFClientDelegate` protocol. The protocol consists of the following methods

```
/**
 Protocol implemented by classes wishing to receive
 notifications.
 Notification primarily of new messages and the state of the
 connection to the server.
 */
@protocol DFClientDelegate

/**
 * This method is called when the DFClient tries to connect, if
 the connection is made, isConnected is true
 * @param isConnected
 */
- (void) onConnection:(BOOL) isConnected;

/**
 * This method is called when the DFClient has lost connection
 */
- (void) onLostConnection;

/**
 * This method is called when the Diffusion server has
 terminated the connection (barred)
 */
- (void) onAbort;

/**
 * This method is called when a message has been received from
 the Diffusion server.
 * This method is called as well as any topicListeners that
 might match the topic.
 */
- (void) onMessage:(DFTopicMessage *) message;

/**
 * This method is called on receipt of the ping request
```

```
 * @see DFClient
 * @param message PingMessage
 */
- (void) onPing:(DFPingMessage *) message;

/**
 * This method is called after a send credentials message, and
 the server rejected the credentials
 * @see DFClient
 */
- (void) onServerRejectedConnection;

/**
 * This method is called if the server did not respond to an Ack
 message in time
 * @see TopicMessage
 */
- (void) onMessageNotAcknowledged:(DFTopicMessage *) message;

/**
 The list of DFServerDetails object has been exhausted, and no
 connection can be placed.
 Once this method is called the set of DFServerDetails is reset
 and further connections can be placed. In most simple scenarios
 where
 there is only one DFServerDetails object in the
 DFConnectionDetails object call method [client connect] here.
 @param client DFClient that has exhausted its set of
 DFServerDetails object from the DFClientDetails object.
 */
-(void)onConnectionSequenceExhausted:(DFClient*)client;

@optional

/**
 Conveys news from the Diffusion server that the named topic no
 longer exists
 */
-(void)onTopicRemoved:(NSString*) topicName;

/**
 The given DFServerDetails object has been selected for
 connection.
 @param details Details object that has been chosen.
 @param client DFClient that has chosen this DFServerDetails
 */
-(void)onConnectionDetailsAcquired:(DFServerDetails*)details
 forClient:(DFClient*)client;
```

You can receive an `onConnection` event after you have successfully connected, this might be because of a lost connection.

Credentials

When credentials are required, use the `credentials` property on the `DFClient` class. Create a `DFCredentials` class and set it on the client before you call `connect`.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `onMessage` method is called on the delegate provided. The message is wrapped in a class called `TopicMessage`. This class contains helper methods that surround the message itself, such as the `topic` and `isInitialLoad` properties. For more information, see iOS Classic API documentation.

Subscriptions

Once the client has connected, you can issue subscribe and unsubscribe commands. The subscribe and unsubscribe methods take a string format, that can be a topic selection pattern, a list of topics that are comma delimited, or a single topic.

Send

Once connected, you can send messages to the Diffusion server on a particular topic path. To do this, use the `send` method.

```
[mClient send:"Fred" :"Hello Fred"];
```

In the example above, the registered message handler for topic Fred receives a `messageFromClient` notification. If you want to send a message with user headers, use the `sendTopicMessage` method. A `TopicMessage` enables you to set user headers.

Ping

You can ping the Diffusion server. The delegate is notified by the `onPing` method. The resulting ping event has two attributes in it, firstly the time stamp of the request. The second attribute is how many items are currently in the client queue at the server. This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the lifetime of the connection, it might be required to have modular components notified about topic messages – these are topic listeners. A topic listener calls a supplied method with a `TopicMessage` class when the topic path of the message matches the topic name.

> **Note:** For performance the iOS topic listeners do not have regular expression patterns but topic name matching.

You can have many topic listeners on the same topic pattern if required. For example, if you want to be notified about a particular topic, issue the following listener:

```
[mClient addTopicListener:aTopicListener];
```

Where a topic listener implements the protocol `DFTopicListenerDelegate` which is shown below.

```
/**
 Protocol for receiving messages from a particular topic.
 */
@protocol DFTopicListenerDelegate

/**
 * This method is called if the TopicMessage matches the message
 received from Diffusion
 *
```

```
 * @param message
 * @return YES if the message is consumed and must not be
 relayed to subsequent DFTopicListenerDelegate, nor the default
 listener.
 */
- (BOOL) onMessage:(DFTopicMessage *) message;

/**
 * This is the topic used to see if the message from Diffusion
 matches (equals) this String
 */
- (NSString *) getTopic;
```

Topic listeners can be removed by calling the `removeTopicListener` method on the `DFClient` class.

iOS examples

Examples that use the iOS Classic API.

F1 Steering Wheel demo

The F1 Steering Wheel demo is available on Github: https://github.com/pushtechnology/blog-steering-wheel/. This demo uses the iOS Classic API to create a display client that shows the real-time input from a driving game controller.

Android Classic API

The Android API is bundled in a library called `diffusion-android-`*`version`*`.jar`, where *version* is the version number, for example 5.5.3.

The Android library is provided with the Diffusion server installation in the `clients/android` folder.

`DiffusionClient` class is the main class that is used. `ConnectionDetails` and the `ServerDetails` classes enables the user to set all of the connection and topic information.

The `DiffusionClient` uses a listener model where the `DiffusionConnectionListener` interface is responsible for all of the event notifications from `DiffusionClient`

Support

Table 70: Supported platforms and transport protocols for the client libraries

Platform	Supported version	Supported transport protocols
Android	2.2	• DPT, DPTS

Features are sets of capabilities provided by the API.

Table 71: Capabilities provided by the API

Feature set	Supported capabilities
Messaging	Send messages through a topic path to the message handler of that topic path

Feature set	Supported capabilities
	Receive messages sent through a topic path
Pings	Ping the Diffusion server
Security	Change the security username and password associated with the client session
Topics	Subscribe to topics Receive updates on subscribed topics Fetch the current state of a topic Unsubscribe from topics

The Diffusion libraries provide capabilities that are not exposed through their APIs. Some of these capabilities can be configured.

Table 72: Capabilities provided by the library

Library capability	Supported
Connect with multiple server details	✓
Transport cascading	✓
Connection failover	✓
Connect through load balancer	✓
Reconnect	✓

Getting started with Android

Create a client application within minutes that connects to the Diffusion server.

About this task

The example demonstrates an empty Android client that you can base your clients on.

Procedure

1. Download the Android SDK and the Diffusion Android library.
2. Create an Android project that references the `diffusion-android-`*`version`*`.jar` library, where *`version`* is the version number, for example 5.5.3.
3. In your project, create a Java file that extends the `android.app.Activity` class.

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

import com.pushtechnology.android.diffusion.DiffusionClient;
import com.pushtechnology.mobile.APIException;
import com.pushtechnology.mobile.ConnectionDetails;
import com.pushtechnology.mobile.DiffusionConnectionListener;
import com.pushtechnology.mobile.DiffusionTopicStatusListener;
import com.pushtechnology.mobile.MalformedURLException;
import com.pushtechnology.mobile.Message;
import com.pushtechnology.mobile.PingMessage;
import com.pushtechnology.mobile.ServerDetails;
import com.pushtechnology.mobile.ServiceTopicError;
import com.pushtechnology.mobile.ServiceTopicHandler;
import com.pushtechnology.mobile.ServiceTopicListener;
import com.pushtechnology.mobile.ServiceTopicResponse;
import com.pushtechnology.mobile.TopicListener;
import com.pushtechnology.mobile.TopicMessage;
import com.pushtechnology.mobile.URL;
import com.pushtechnology.mobile.enums.EncodingValue;

public class DemoClient extends Activity implements
 DiffusionConnectionListener, DiffusionTopicStatusListener {

  private DiffusionClient theClient;
  static final String TAG = "Diffusion Client";
  private static final String SERVICE_TOPIC = "SERVICE";

  ConnectionDetails cnxDetails;
  {
    try {
      ServerDetails svrDetailsArr[] = new ServerDetails[] {
          new ServerDetails( new URL( "URL:port" ) ),
      };
      cnxDetails = new ConnectionDetails( svrDetailsArr );
      cnxDetails.setTopics( SERVICE_TOPIC );
      cnxDetails.setCascade( true );
      cnxDetails.setAutoFailover( true );
    } catch (MalformedURLException ex) {
      writeln( ex.toString() );
    }
  }
  ServerDetails currentServerDetails;


  // Set Diffusion connection details, and place the Diffusion
 connection
  private void connectToServer()
     {
    theClient = new DiffusionClient();
    theClient.setConnectionDetails( cnxDetails );
    theClient.addTopicListener(new TopicListener());
    theClient.setConnectionListener(this);
    theClient.setTopicStatusListener( this );
    theClient.connect();
    statusText.setText( String.format( "Connecting to %s",
 currentServerDetails.getUrl() ) );
  }
```

```
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    // Set up the UI of your Android app

    connectToServer();
  }

}
```

a) Import `com.pushtechnology.android.diffusion.DiffusionClient` and the classes in the `com.pushtechnology.mobile` package.
b) Implement `DiffusionConnectionListener` and `DiffusionTopicStatusListener`.
c) Create a `ConnectionDetails` object that includes the connections details and settings that you require.
d) Create a `connectToServer()` method that uses the `ConnectionDetails` object to set the connections details and places a connection to Diffusion.
e) Override the `onCreate()` method inherited from `Activity`. In this method call `connectToServer()` to place the Diffusion connection when the application starts.
f) Use the `onCreate()` method to set up the views and content of your Android application.

Related Links

http://developer.android.com/training/index.html

Using the Android Classic API

There are features, issues, and considerations that are specific to clients that are implemented using the Android Classic API.

Credentials

When credentials are required, there are three ways to set the credentials. The `ServerDetails`, the `ConnectionDetails` and the `DiffusionClient` all have a `setCredentials` method. It is required that the user create a `DiffusionCredentials` object and pass it to one of these methods before calling connect. Use only one of these ways. If more than one way is used, the selection of the credentials to use is undefined. The Android Classic API only supports sending credentials on connection.

onMessage event

When messages arrive from the Diffusion server on a subscribed topic, the `onMessage` is called on the delegate provided. The message is wrapped in a interface called `Message`. This interface contains helper methods that surround the message itself, like `getTopic()` and `isInitialTopicLoad`. For more information, see Android Classic API documentation.

Subscriptions

Once the client has connected, you can issue subscribe and unsubscribe commands. The `subscribe` and `unsubscribe` methods take a string format, that can be a topic selector pattern, a list of topics that are comma delimited, or a single topic.

Send

Once connected a client can send messages to the Diffusion server on a particular topic. To do this, use the `send` method.

```
theClient.send("Fred","Hello Fred");
```

In the example above, the registered message handler of the topic Fred receives a `messageFromClient` notification. If the message requires a user header, use the `sendTopicMessage` method. A `TopicMessage` allows for the setting of user headers.

```
    TopicMessage message = new TopicMessage("Fred");
    message.addUserHeader("myHeaders");
    message.setMessage("Hello Fred");
    theClient.sendTopicMessage(message);
```

Ping

The client can ping the Diffusion server. The delegate is notified of the response by the `onPing` function. The resulting ping event has two attributes in it:

- the timestamp of the request
- the number of items in the client queue

This information enables the client to get some vital connection information. It is down to the implementation of the client to specify the ping frequency, if at all required.

Topic listeners

During the life time of the connection, it might be required to have modular components that get notified about topic messages, these are topic listeners. A topic listener is called using its `onMessage` method with a message object when the topic of the message matches the topic name.

> **Note:** For performance the Android topic listeners do not have regular expression patterns but topic name matching.

You can have many topic listeners on the same topic path if required. For example, if you want to be notified about a particular topic, use the following code:

```
theClient.addTopicListener(topicListener);
```

Where `topicListener` implements the interface `TopicListener`. See the following example.

```
   /**
    * getTopic
    *
    * @return the topic that this listener is interested in.
This does
    * not take regular expressions this must be an exact match
    */
   String getTopic();

   /**
    * onMessage
    *
    * @param message message which topic matches the getTopic
method
    */
```

```
    void onMessage(Message message);
```

Remove topic listeners by calling the `removeTopicListener` method on the `DiffusionClient` class.

Threading concerns

The `DiffusionClient` creates and dedicates a thread to listening to traffic from the Diffusion server and reacting to messages from it. Consequently methods on the `DiffusionConnectionListener` and `DiffusionTopicStatusListener` are executed in the same thread. Android does not allow background threads to interact with GUI controls, only the main thread is allowed to do so.

To overcome this, any non-main thread can pass a `java.lang.Runnable` to the main thread for execution via `android.view.View.post(Runnable action)`. For example:

```
/*
 * Called when the Diffusion connection is established
 */
public void connected() {
    // Post this Runnable to the GUI thread to change the
 display
    String statusTextStr = String.format( "Connected to %s:%d
\n", HOST, PORT );
    setStatus( statusTextStr );
    Log.i( TAG, statusTextStr );
  }

   /**
    * Set the content of the status view
    * @param statusStr
    */
   private void setStatus(final String statusStr)
    {
// Pass a Runnable to the GUI thread to execute using one of its
 widgets
     outputView.post( new Runnable() {
         public void run() {
            statusText.setText( statusStr );
        }
     } );
 }
```

User permissions in `Manifest.xml`

To establish a connection with the Diffusion server, Android devices must add the user permission `INTERNET` within the `Manifest.xml`. This permission allows applications to open network sockets.

Android examples

Examples that use the Android Classic API.

DiffusionClient

The following code shows an example of connection:

```
    // Get a new DiffusionClient
    theClient = new DiffusionClient();

    //Set the connection details
```

```
    ServerDetails serverDetails = new ServerDetails(new
URL("dpt://diffusion.example.com:80"));
    ConnectionDetails connectionDetails = new
ConnectionDetails(serverDetails);
    theClient.setConnectionDetails(connectionDetails);

    // Make this listen to the DiffusionClient events
    theClient.setConnectionListener(this);

    // Connect
    theClient.connect();
```

DiffusionConnectionListener

The DiffusionConnectionListener interface consists of the following methods (for further information refer to the API documentation)

```
    /**
     * connected, called upon connection
     */
    void connected();

    /**
     * errorConnecting, called if there is an error connecting
     *
     * @param e
     */
    void errorConnecting(Exception e);

    /**
     * disconnected, called when the connection list lost
     */
    void disconnected();

    /**
     * connectionAborted, called when DiffusionServer has
rejected the connection
     */
    void connectionAborted();

    /**
     * onMessage, called when a message has been received from
Diffusion
     *
     * @param message
     */
    void onMessage(Message message);

    /**
     * onPingMessage, called when a ping response is received
     *
     * @param message
     */
    void onPingMessage(PingMessage message);

    /**
     * onMessageNotAcknowledged, called when an ack message has
not been acknowledged by Diffusion
     *
     * @param message
     */
```

```
    public void onMessageNotAcknowledged(TopicMessage message);


    /**
     * onConnectionSequenceExhausted, called when the complete
list of ServerDetails have been exhausted.
     */
    public void onConnectionSequenceExhausted();

    /**
     * onConnectionDetailsAcquired, called each time a
ServerDetails object is selected for connection.
     *
     * @param serverDetails
     */
    public void onConnectionDetailsAcquired(ServerDetails
serverDetails);

    /**
     * onServerRejectedCredentials, called when Diffusion reject
the credentials.
     */
    public void onServerRejectedCredentials();
```

Change the URL from that provided in the example to the URL of the Diffusion server.

C API

The C API is provided as a source distribution in the file `diffusion-c-classic-`*`version`*`.zip`, where *`version`* is the version number, for example 5.5.3. This file is located in the `clients/c` directory of your Diffusion installation.

The source builds to either a dynamic or shared library on UNIX systems and is supported on Red Hat 7.1 and CentOS 7.1. You can link it into your own C/C++ applications, or used as the foundation for creating Diffusion clients for other languages which can be extended through binary APIs.

Using the C API

Build the library using the make command and ensure that it is on your LD_LIBRARY_PATH.

Building

To build the library, type `make` in the source directory. This builds shared and static versions of the Diffusion library, `libdiffusion.so` and `libdiffusion.a` respectively. Additionally, a number of sample applications are also built.

Installation

Copy the libraries to a location on the user's `LD_LIBRARY_PATH` (or equivalent). Copy the header files, `diffusion.h` and `llist.h` to your C compiler's include path.

For example, on Ubuntu® 10.10 the following is appropriate:

```
 # mkdir /usr/local/include/diffusion
 # cp diffusion.h llist.h /usr/local/include/diffusion
 # mkdir /usr/local/lib/diffusion
 # cp libdiffusion.* /usr/local/lib/diffusion
 # echo "/usr/local/lib/diffusion" > /etc/ld.so.conf.d/
libdiffusion.conf
 # ldconfig
```

Example usage

The following example shows how to connect to a Diffusion instance (no credentials):

```
DIFFUSION_CONNECTION *c = diff_connect("localhost", 8080,
NULL);
if(c == NULL) {
 fprintf(stderr, "Failed to connect to Diffusion\n");
 return(-1);
}
```

The following example shows how to connect to a Diffusion instance (with credentials)

```
SECURITY_CREDENTIALS creds;
creds.username = strdup("smith");
creds.password = strdup("secret");
DIFFUSION_CONNECTION *c = diff_connect("localhost", 8080,
&creds);
if(c == NULL) {
 fprintf(stderr, "Failed to connect to Diffusion\n");
 return(-1);
}
```

The following example shows how to request a subscription to a topic:

```
DIFFUSION_CONNECTION *c = diff_connect(...);

if(diff_subscribe(c, "Assets") == -1) {
 fprintf(stderr, "Failed to subscribe to topic\n");
 return(-1);
}
```

The following example shows how to use the event loop and callbacks:

```
void on_initial_load(DIFFUSION_MESSAGE *msg) {...}
void on_delta(DIFFUSION_MESSAGE *msg) {...}

...

DIFFUSION_CONNECTION *c = diff_connect(...);
diff_subscribe(...);

DIFFUSION_CALLBACKS callbacks;
DIFF_CB_ZERO(callbacks);  // Reset callback structure
callbacks.on_initial_load = &on_initial_load;
callbacks.on_delta = &on_delta;

diff_loop(c, &callbacks);
```

diffusion-wrapper.js

The Diffusion wrapper is a script which addresses a weakness in the Flash and Silverlight VMs – when running inside a web browser there is no provision for the execution of callback code when the user closes either the containing tab or the entire browser window.

Consequently there is no opportunity for the Diffusion client to inform the server that the client is willingly closing the connection, instead the connection is severed. This can result in various server warnings (dependent on the transport mechanism, that is DPT or HTTP) as well as maintaining the server-side reference to the client if any keep-alive properties are set.

The JavaScript environment in the hosting browser provides browser closing callbacks, and these are employed by DiffusionWrapper.js. By supplying the `setClientDetails` function with client and server info after a connection has been established within the Flash or Silverlight API, DiffusionWrapper attaches a JavaScript function to the `onbeforeunload` event that is triggered when a browser window or tab closes. This notifies Diffusion that the client is deliberately closing.

`diffusion-wrapper.js` can be found in `clients/flex` and `clients/silverlight` directories from a default installation.

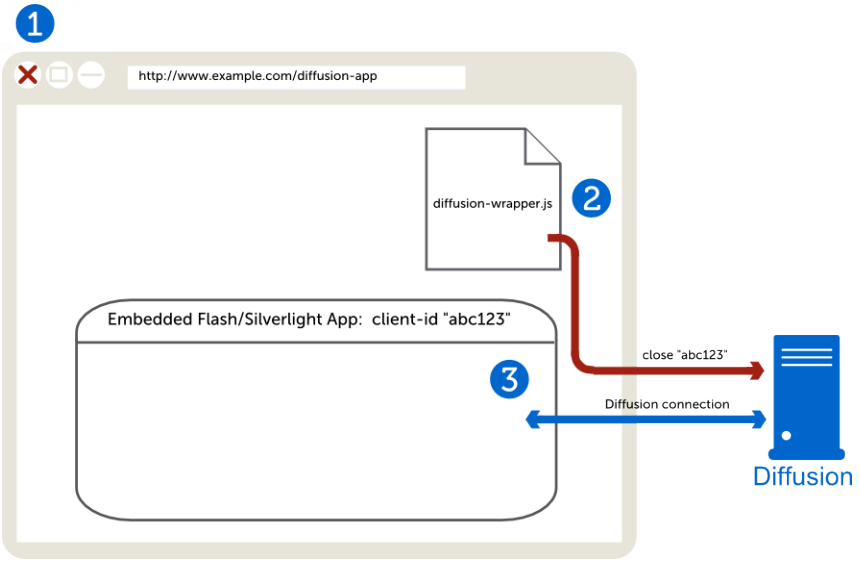


Figure 41: Diffusion wrapper

The preceding diagram shows `diffusion-wrapper.js` in use. When the user closes the containing browser tab or window, the following events occur:

1. The user closes the browser. This engages a JavaScript callback that calls `diffusion-wrapper.js`.
2. `diffusion-wrapper.js` runs and sends a closure request to the Diffusion server.
3. The Flash or Silverlight client dies. It has no chance to run cleanup code.

How to use Diffusion wrapper

The HTML page that loads the Flash/Silverlight app, must to load the Diffusion wrapper script.

The following example shows the `diffusion-wrapper.js` file being included in a web page:

```
<html>
 <head>
  <script src="diffusion-wrapper.js" language="javascript"></
script>
 </head>
 <body>
 .....
 </body>
</html>
```

The Flash/Silverlight app makes a call to the external DiffusionWrapper method `setClientDetails`, typically in the `onConnection` callback, supplying the client ID and server URL. In the example below, the method `ExternalInterface.call("setClientDetails")` provides DiffusionWrapper with

the client ID and server URL. DiffusionWrapper adds a method to the `onbeforeunload` event of the window, which informs the Diffusion server that the client is intentionally closing the connection when the browser window or tab is closed.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                width="371" height="334" minWidth="955"
                minHeight="600" layout="absolute">
<mx:Script>
 import com.pushtechnology.diffusion.ConnectionDetails;
 import com.pushtechnology.diffusion.DiffusionClient;
 import com.pushtechnology.diffusion.ServerDetails;
 import
 com.pushtechnology.diffusion.events.DiffusionConnectionEvent;
 import
 com.pushtechnology.diffusion.events.DiffusionExceptionEvent;
 import
 com.pushtechnology.diffusion.events.DiffusionMessageEvent;


 private var theServerUrl:String = "http://127.0.0.1:8080"
 private var theInitialTopic:String = "Echo"
 private var theClient:DiffusionClient;

 private function onConnect():void{
     theClient = new DiffusionClient();
     var theConnectionDetails:ConnectionDetails = new
 ConnectionDetails(new ServerDetails(theServerUrl),null);

     // Add the listeners

 theClient.addEventListener(DiffusionConnectionEvent.CONNECTION,
 onConnection);
     theClient.addEventListener(DiffusionMessageEvent.MESSAGE,
 onMessages);

 theClient.addEventListener(DiffusionExceptionEvent.EXCEPTION,
 onException);

     //Lets Go...
     theClient.connect(theConnectionDetails);
 }

 private function
 onConnection(event:DiffusionConnectionEvent):void{
     // Is the client connected?
     if(event.isConnected()){
         //Check the externalInterface availability
        if(ExternalInterface.available){
          // Send the ClientId and serverURL to the
 DiffusionWrapper
    ExternalInterface.call("setClientDetails",
 theClient.getClientID(), event.getServerDetails().getURL());
         }else{
         //The externalInterface is not available, so the
 DiffusionWrapper is not called.
        }
   }
 }

 private function onMessages(event:DiffusionMessageEvent):void{
  //Message received
```

```
        theMessages.text += event.toString();
    }

    private function
    onException(event:DiffusionExceptionEvent):void{

     //Exception received
     theMessages.text += event.toString();
    }

</mx:Script>
    <mx:TextArea id="theMessages" x="18" y="40" width="335"
    height="283" text="Messages"/>
    <mx:Button id="bConnect" x="105.5" y="10" width="160"
    label="Connect" click="onConnect()"/>
</mx:Application>
```

Testing

This section covers some aspects of testing a Diffusion system.

Flex/Flash client

The Flex client test tool is a Flex application that uses the diffusion-flex.swc library. The tool can be found in the Tools section in the root menu of your Diffusion installation.

This tool enables the user to supply connection information as well as credential information. Select which transport is going to be used to connect to the Diffusion server. This client can issue a server ping. The ping response displays the elapsed time, its average and the current queue size for this client.

This tool can also be used to test reconnection, auto failover, and load balancing. Available servers can be listed by IP in the URL box. The **Reconnect** button, after a disconnect, returns the client to the server it was originally connected to, and receive messages queued during disconnection. Selecting **Auto Failover** causes the client to connect to the next available server, after the existing connection is forcibly killed. Selecting **Cascade** causes the client to cascade through the available servers until it finds one to which it can connect. Select **Load Balancing** in conjunction with **Auto Failover**, and the client connects to one of a shuffled list of available servers.

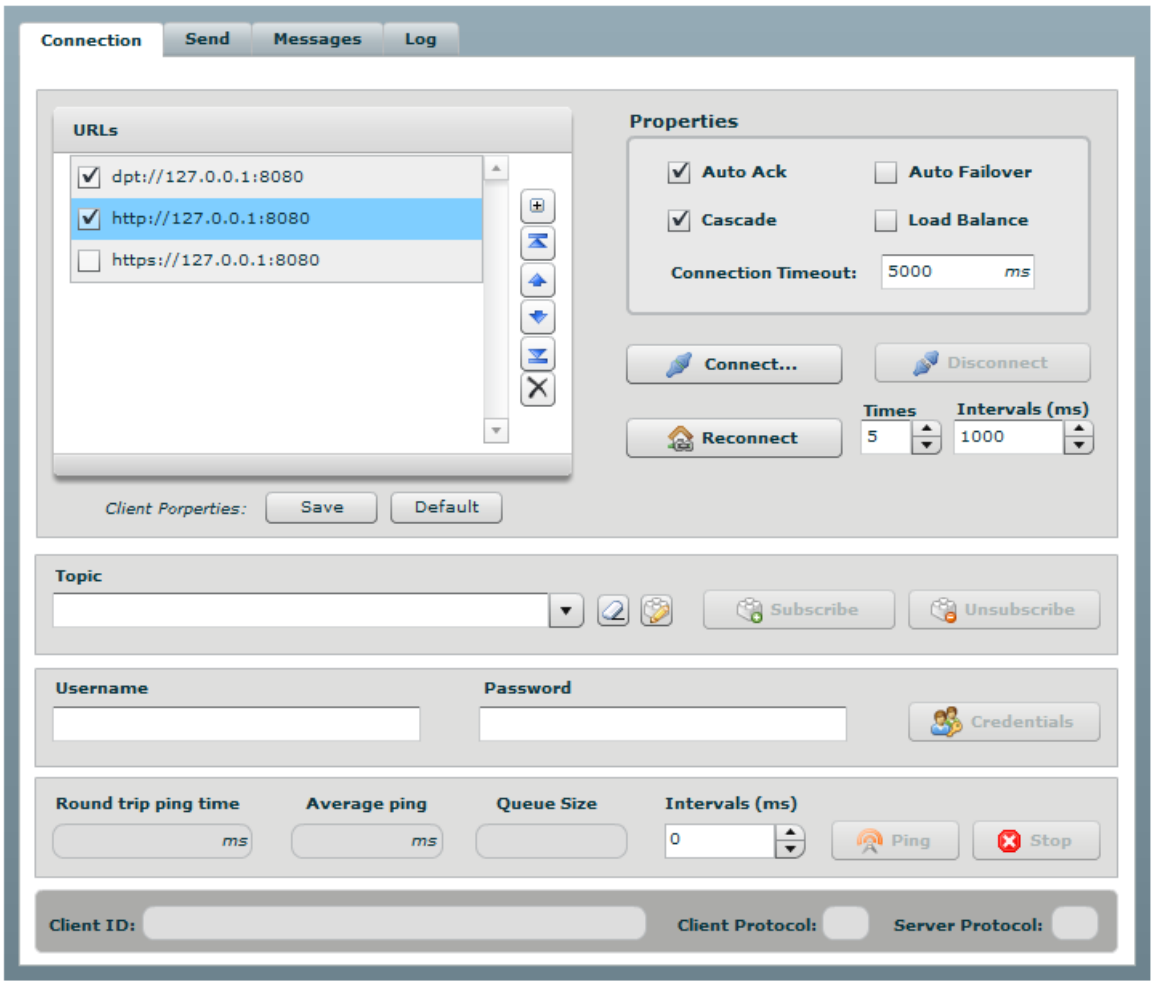


Figure 42: Flex client: Connection tab

You can pass user headers with a fetch request which are reflected back to the client in the response message as a Correlation ID. This allows the client to be able to associate replies with requests. The Flex client API has a new 'headers' parameter on the fetch methods. No changes are required to publishers to benefit from this.

Once connected to a Diffusion server, you can send messages to any topic. The publisher receives this message and notify the `messageFromClient` method in the publisher. The requested encoding can be set, as well as any user headers required. You can request an acknowledgment from the server.

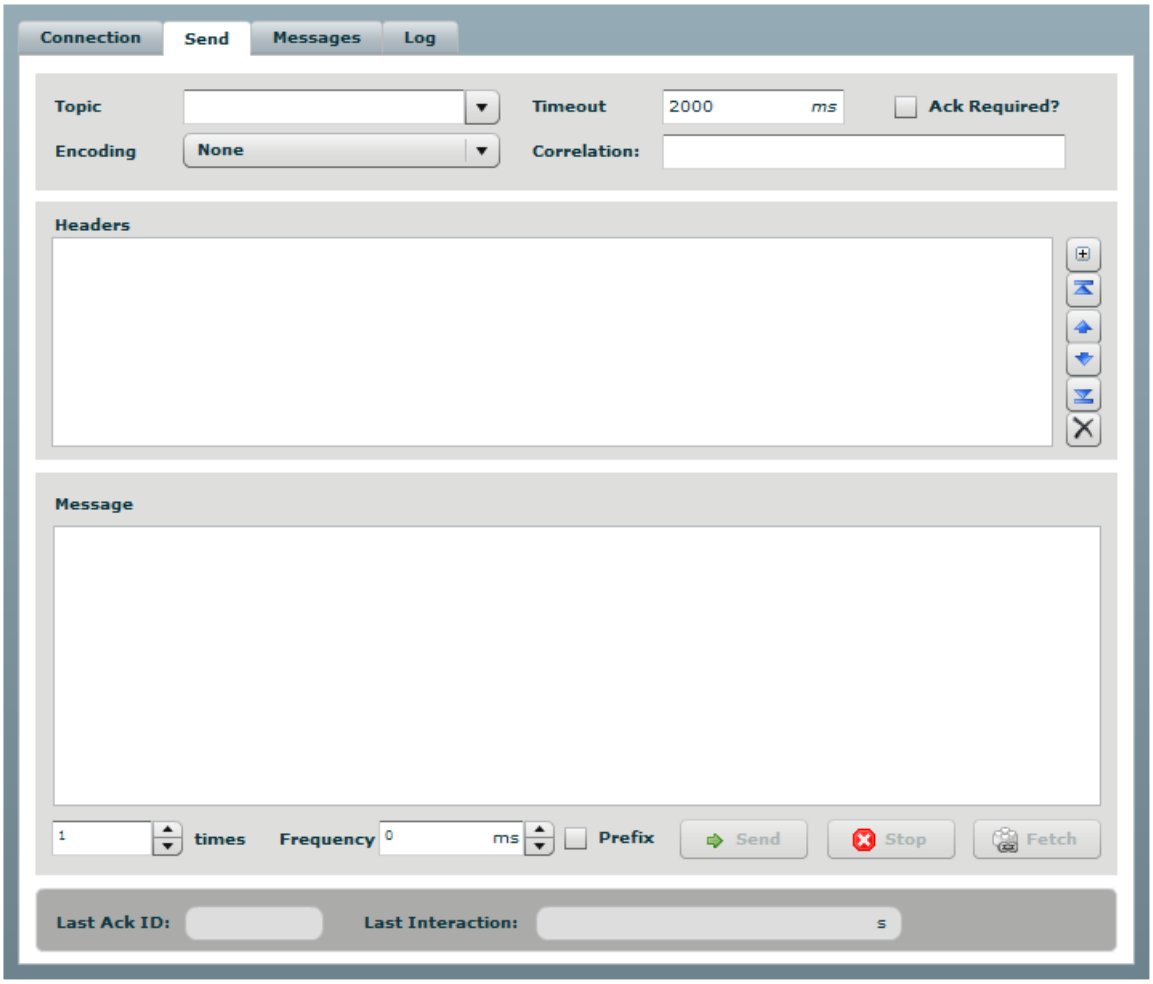


Figure 43: Flex client: Send tab

Messages are displayed in a tree format. It is not a true hierarchic tree, as each topic has its own node directly off the root node.

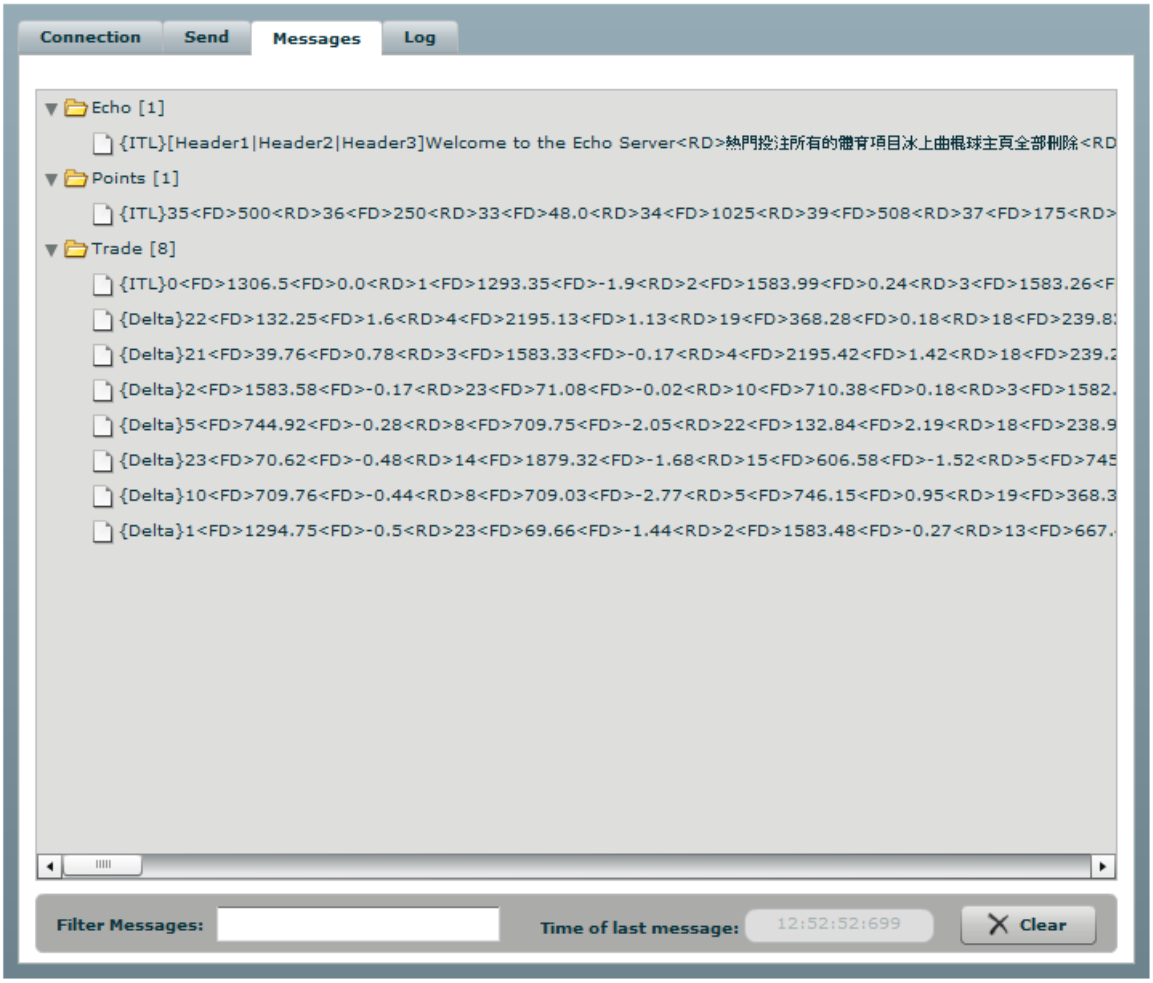
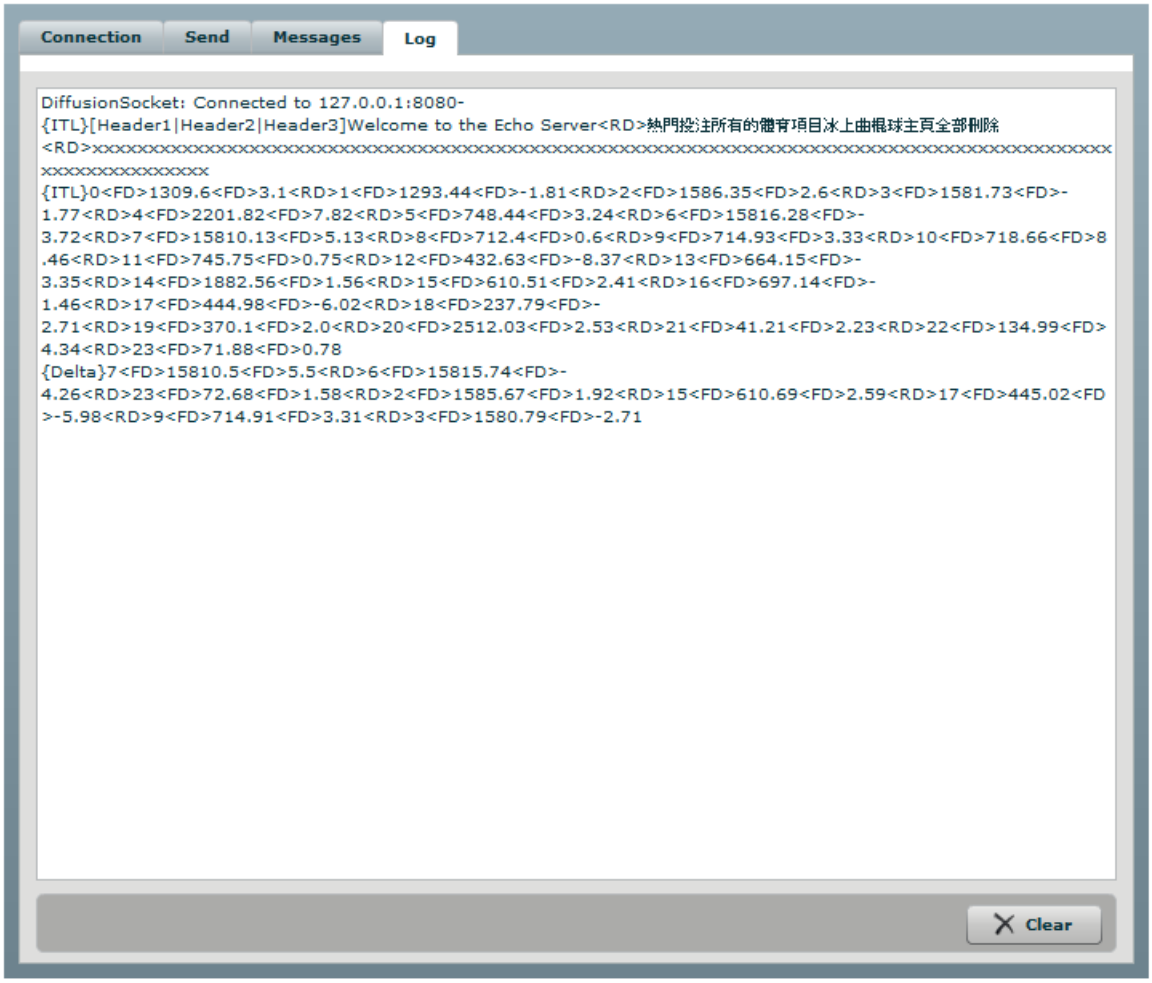


Figure 44: Flex client: Messages tab

The log tab shows messages that are sent from the Diffusion client when debugging is switched on.


Connection
Send
Messages
Log
DiffusionSocket: Connected to 127.0.0.1:8080-
{ITL}[Header1|Header2|Header3]Welcome to the Echo Server<RD>熱門投注所有的體育項目冰上曲棍球主頁全部刪除
<RD>xxxx

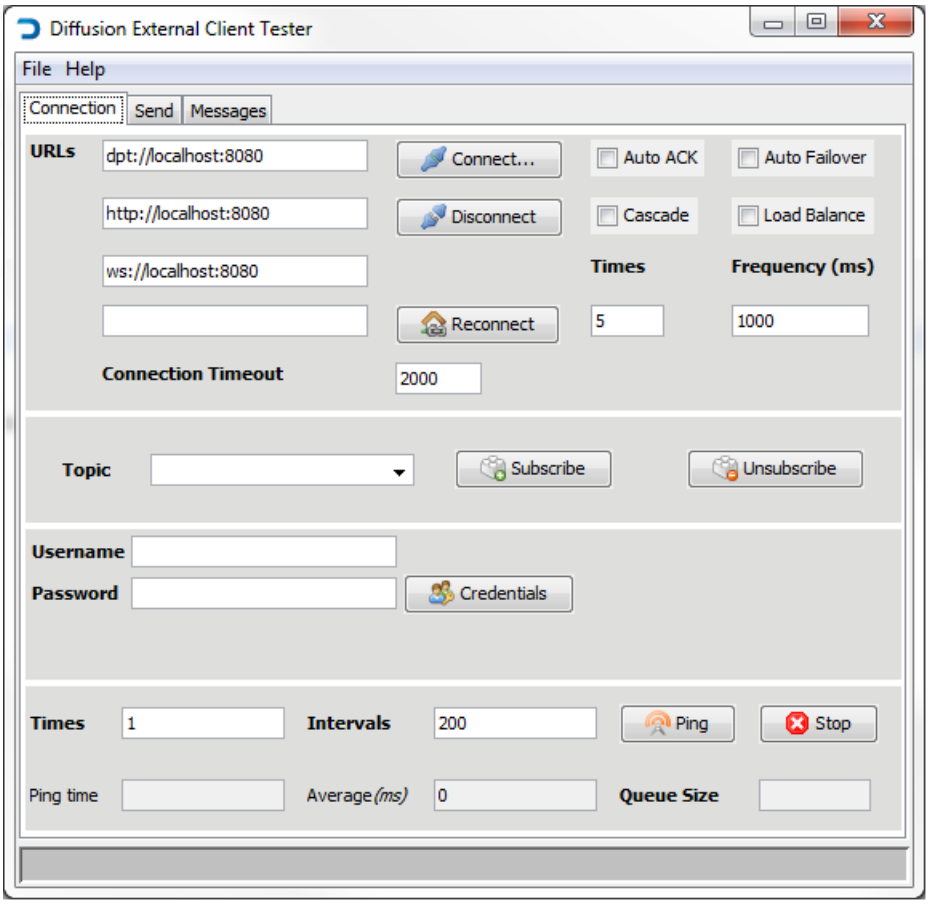


Figure 46: External client tester: Connection tab

You can connect to Diffusion securely. In order for the external client to work in secure mode, a certificate must be installed on the client machine. To aid in installing the certificate, there is an option under the file menu called Load Cert. This connects to Diffusion and detail instructions on how to install the newly created jssecerts file.

After subscribing to a topic, the client can send messages to that topic. The publisher receives this message and notify the `messageFromClient` method in the publisher. It is also possible to set the message encoding type and any user headers if required.

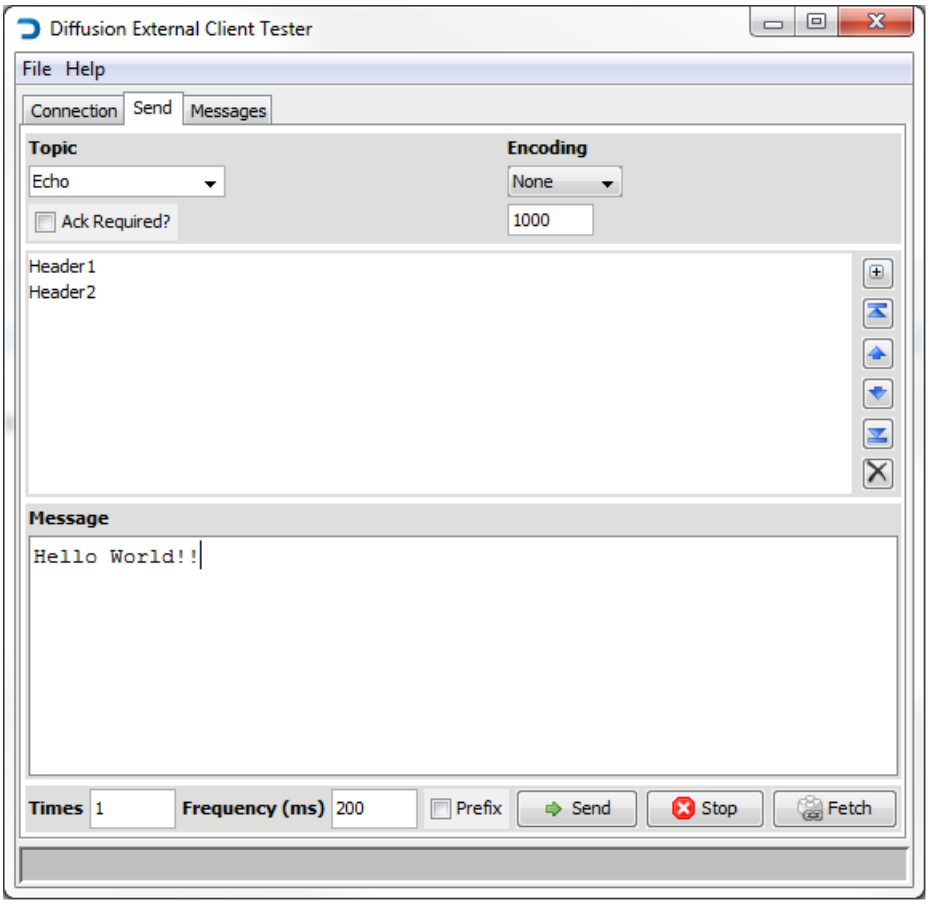


Figure 47: External client tester: Send tab

Messages are displayed in a tree format. It is not a true hierarchic tree, as each topic has its own node directly off the root node.

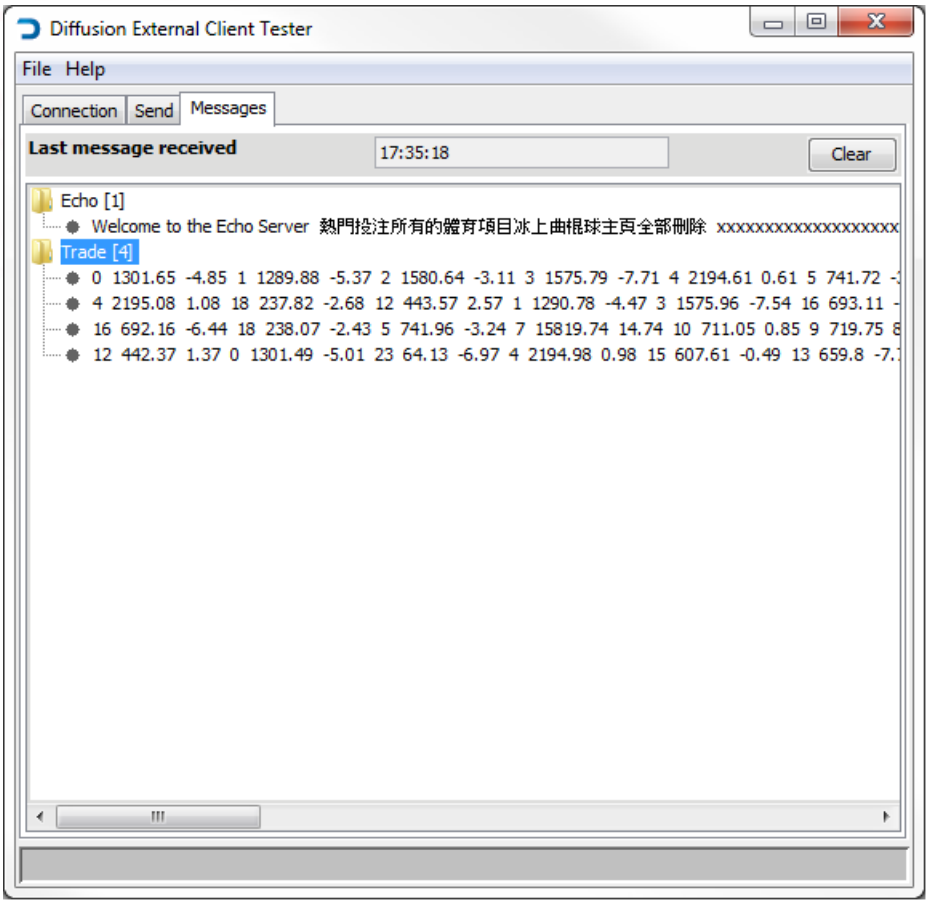


Figure 48: External client tester: Messages tab

It is also possible to examine a message by double clicking it

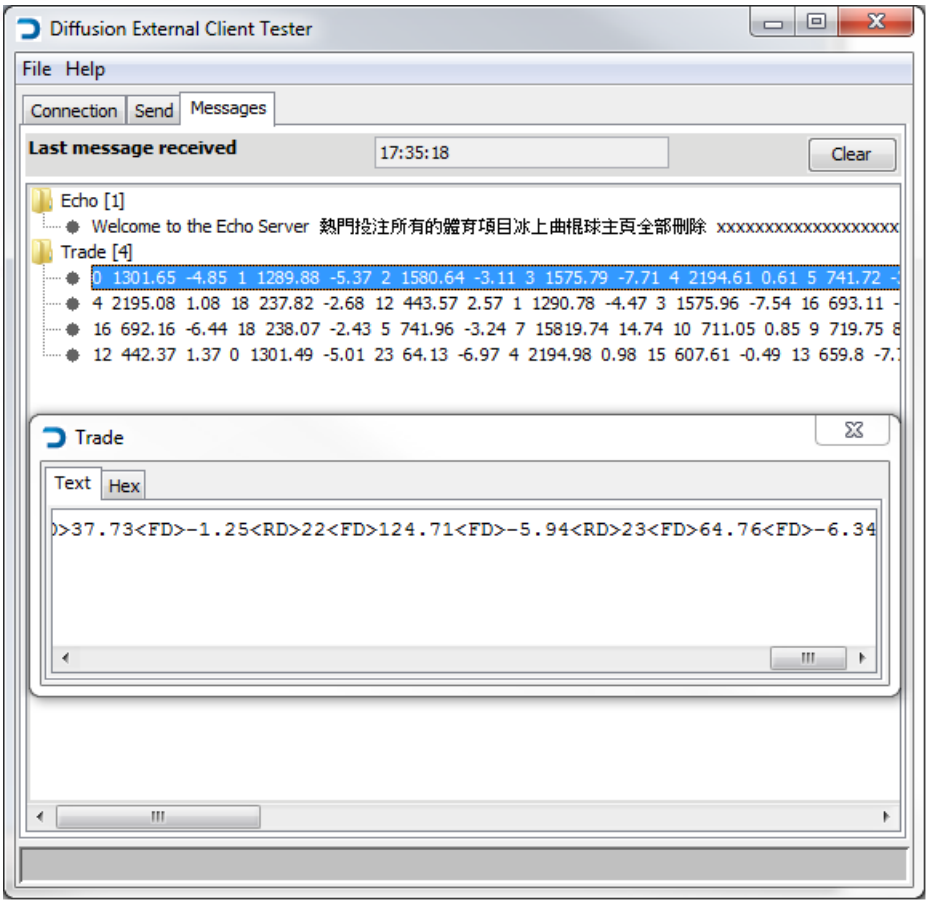


Figure 49: External client tester: Message details window

JavaScript client test tool

The JavaScript test tool allows the user to connect to a publisher and test the API.

The JavaScript test tool is browser based and can be found in the Root Menu of your Diffusion installation. It allows for different types of connections. The test tool also allows for Subscribing, Unsubscribing and Fetching against topic sets.

Fetch Correlation allows user headers to be passed with a fetch request which is reflected back to the client in the response message. This allows the client to be able to associate replies with requests. The JavaScript client API has a new headers parameter on the fetch methods. See API documentation for changes to the API specification. No changes are required to publishers to benefit from this.

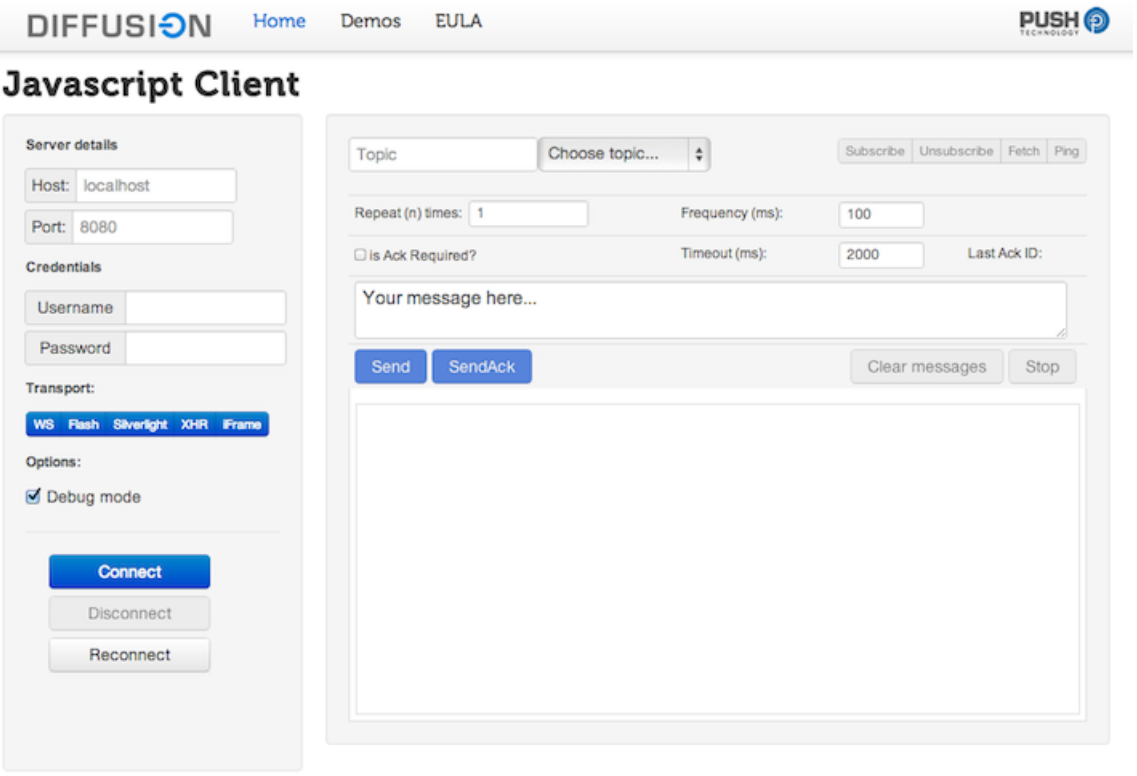


Figure 50: JavaScript test tool

Silverlight client test tool

The Silverlight client test tool is a Silverlight application that uses the `PushTechnology.Transports` assembly. This tool enables the user to perform various diagnostic functions without having to build a new Silverlight application.

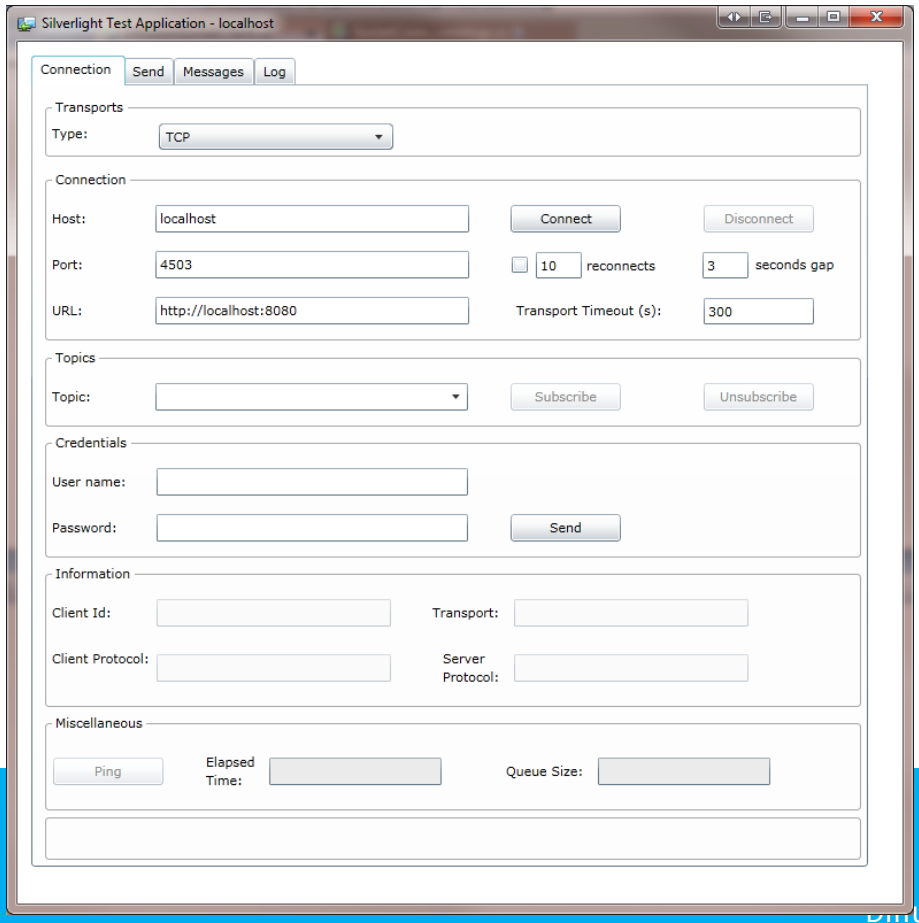


Figure 51: Silverlight test tool: Connection tab

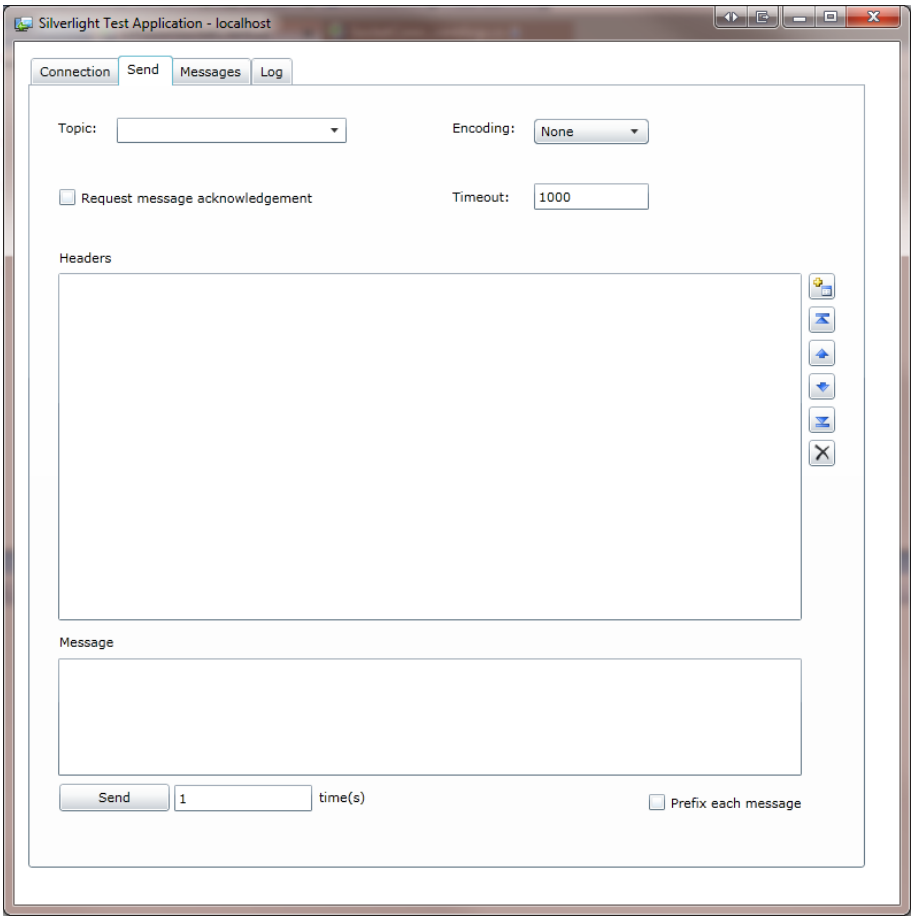


Figure 52: Silverlight test tool: Send tab

Received messages are displayed in a hierarchical format:

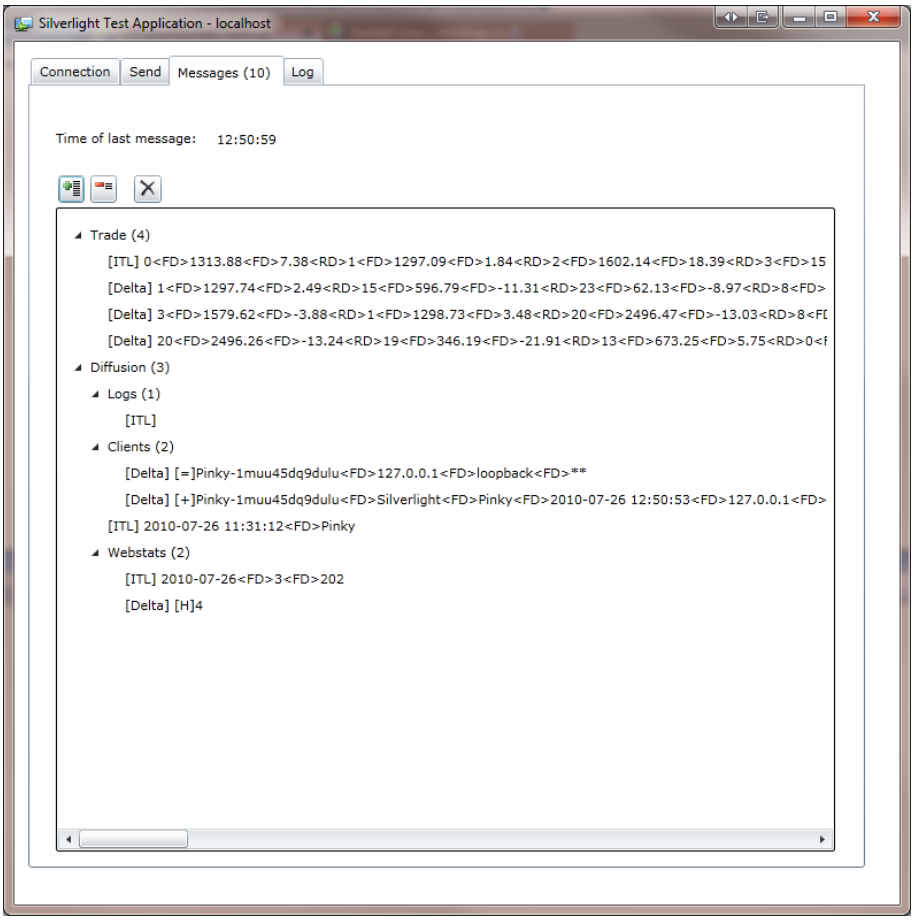


Figure 53: Silverlight test tool: Messages tab

Windows client test tool (.NET)

The external client test tool is a .NET application that enables the user to connect/subscribe/unsubscribe to a Diffusion server. It allows for many different connection types (for example, HTTP/DPTS), and also allows for the composition and sending of messages.

Below is the **Connection** tab of the external client test tool; this contains all of the facilities required to establish a connection to the Diffusion server.

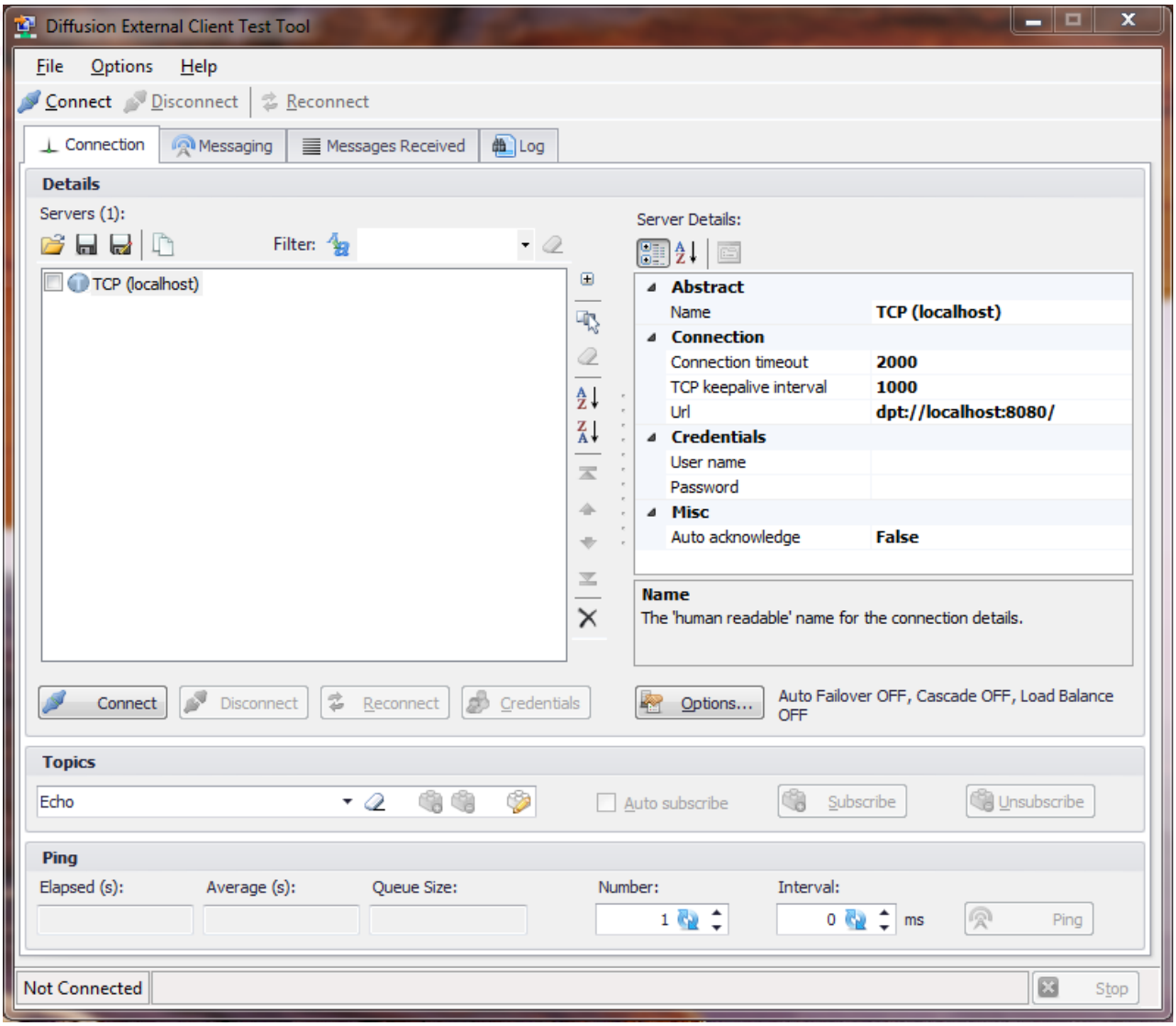


Figure 54: Windows client test tool: Connection tab

Additional topics can be subscribed at any time subsequent to connection by entering the topic names in the topics field and clicking the **Subscribe** button. The converse action (unsubscription) can be performed with the **Unsubscribe** button.

You can issue a server ping from this client. The elapsed time is indicated in the relevant fields, along with a rolling average.

If connection credentials are required to subscribe to a particular topic, these can be entered in the **Credentials** section. They can also be sent retrospectively using the **Credentials** button.

Various connection options are available to the user by clicking the **Options** button:

Auto failover

After a connection to Diffusion is placed, if the connection fails and if there is a pool of server details to pick from, the next set of server details is chosen and connection placed to the next server.

Cascade

When the client software attempts to place a connection, if the connection attempt fails, the next set of server details in the list is chosen and used. It is similar to Auto Failover except this logic is applied prior to a connection, whereas Auto Failover is applied once a connection is in place.

Load balance

If **Load Balance** is set, the list of server details is shuffled prior to use.

Restore state when reconnecting

If connection to Diffusion is lost, and the server has been configured accordingly, the client reconnects and receives any messages that have been missed when the connection failed.

Once subscribed to a topic, you can send messages to either that specific topic, or any registered topic, using the **Send** tab:

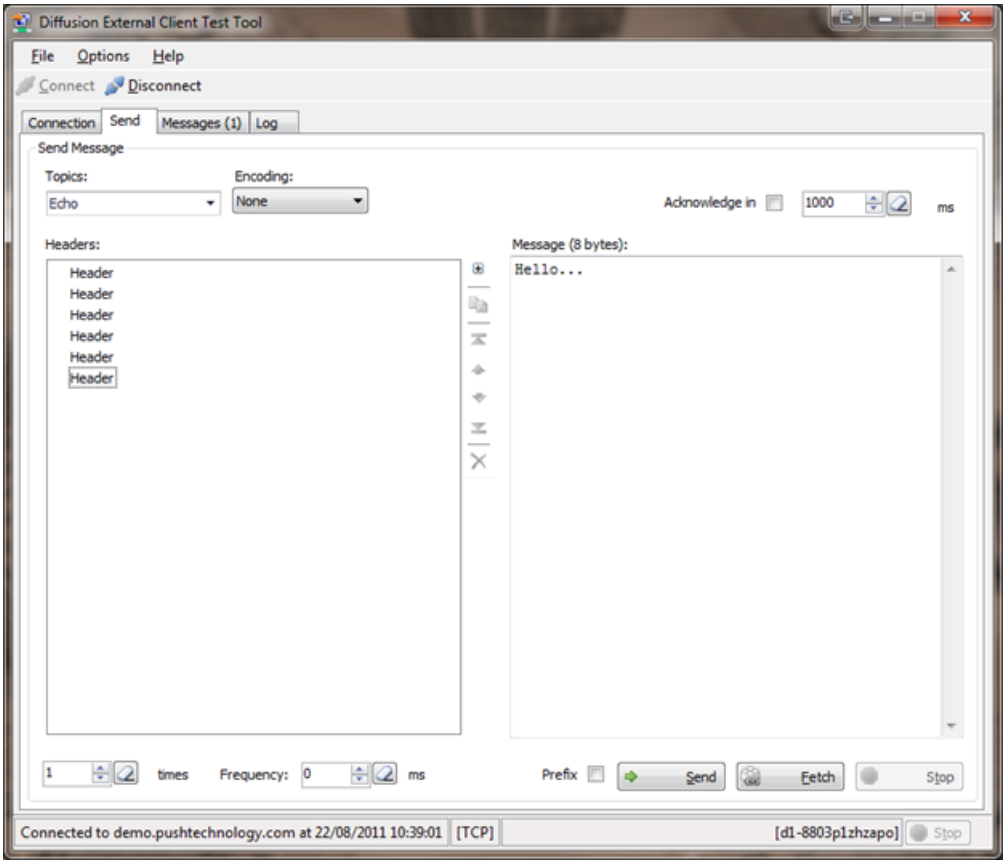


Figure 55: Windows client test tool: Send tab

As can also be seen, you can compose a message containing headers; these can be added using the convenient toolbar to the side of the main headers area. You can also send a message multiple times by incrementing the edit field at the bottom of the message composition area.

Any received messages from the Diffusion server are shown in the **Messages** tab as shown in the following figure:

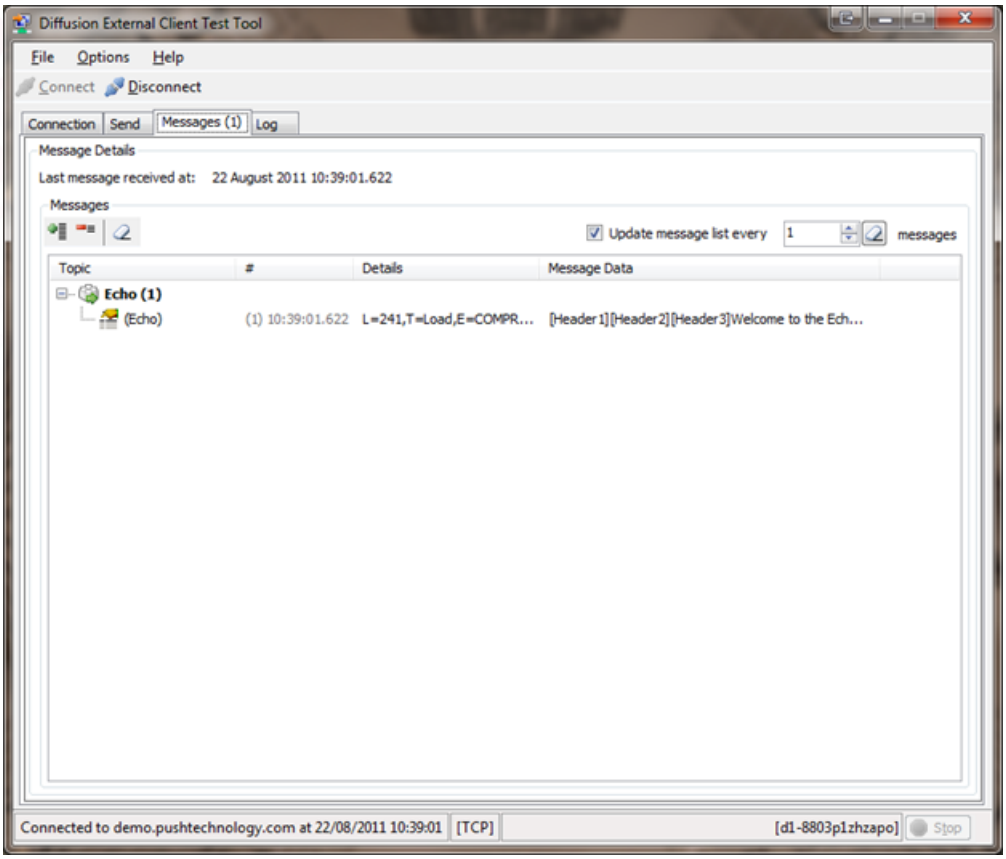


Figure 56: Windows client test tool: Messages tab

Messages are displayed in a true hierarchical format, so deeply-nested topics will be relatively easy to locate and analyze.

If a high volume of messages is expected from the Diffusion server, you can filter the number of messages by adjusting the **Update message list** value. You can disable message updates completely by clicking the check box.

You can view logging information generated by the .NET libraries:

The logging type and level can be adjusted using the **Preferences** option on the main menu.

Stress test tuning

Stress testing Diffusion requires various tuning changes to be made.

Table 73: Tuning changes for stress testing

Increase number of open files	If the Diffusion server is running in Linux, there is a limit to the number of open files/ connections. See "Increasing Number of Open files".
Buffer sizes	Socket buffer sizes can be changed in `etc/Connectors.xml`
Inbound thread settings	The Inbound thread pool settings must be tuned otherwise the pool and queues get full.

	The settings are found in `etc/Server.xml`. Increase these values carefully and the server re-tested each time. Set the maximum size and queue size quite high for stress testing (for example, 250) so that the server can handle the hundreds or thousands of simultaneous connection attempts.
Client queues	Depending on the number and frequency of the messages, you might have to increase the client queue parameter.
Client multiplexers	By default, the number of client multiplexers configured is equal to the number of cores on the host system of the Diffusion server. You might have to increase the number of multiplexers as the client load on each Diffusion server increases.

Limitations

If more than 2000 clients are to be simulated, it is better to run the stress test from multiple machines. Most systems start to struggle when dealing with 2000 concurrent connections. By running the stress test on multiple machines any number of clients can be tested.

Socket exceptions can be encountered when a large number of clients are run on one machine. These can be thrown on either the Diffusion server or within the stress test tool. If this is occurring, the number of clients needs to be reduced

Stress test

Stress test is a good way of working out what the system requirements are going to be of a Diffusion server under duress. The stress test tool is issued to test the performance of Diffusion under stress and helps to understand how changes to the configuration can have an effect upon the performance.

Before you begin

Do not run the stress test tool on the same machine as the Diffusion server. A separate machine is required to run the stress test tool.

About this task

Installing the stress test tool;

Procedure

1. On the test machine, create a `stresstest` directory and copy the files `stresstest/stresstest.bat`, `stresstest/stresstest.sh`, `stresstest/stresstest.properties` and `lib/diffusion.jar` from the Diffusion installation
2. Configure stress test by editing the `stresstest.properties` file. The following need to be configured:
3. Number of clients
4. Transport type
5. Server details
6. Number of messages with which to run the test

7. The topic to which the clients subscribe
8. On a Windows system, launch `stresstest.bat`, on a UNIX machine launch `stresstest.sh`.

Benchmarking suite

A benchmarking suite for Diffusion is available on GitHub. You can use this suite to test the latency and throughput of publishers.

The benchmarking suite is available at the following location: https://github.com/pushtechnology/diffusion-benchmark-suite.

The benchmarking suite works on Linux only and requires the following software be installed on the system:

- Apache™ Ant™
- Java with JDK
- Diffusion server

For more information about using the benchmarking suite, see the readme file in the GitHub project.

Test tools

Test tools are provided which allow you to connect to a Diffusion server as an external client application. There are also tools that act as a publisher server application.

Generic Java versions are provided which work on any platform and Windows versions are provided specifically for use on Windows platforms.

Table 74: Testing tools

Java client	Provides a GUI interface simulating an external client connection using the Java API. Suitable for use on any platform.
Windows client	Provides a GUI interface simulating an external client connection using the Windows API. Suitable for use on Windows platforms only.
JavaScript client	Provides a GUI interface simulating a JavaScript client connection. Suitable for use on any platform.
Flex/Flash client	Provides a GUI interface simulating a Flash client connection. Suitable for use on any platform where Flash Player is installed.
Silverlight client	Provides a GUI interface simulating a Silverlight client connection. Suitable for use on any platform where Silverlight is installed.

Part IV

Administrator Guide

This guide describes how to deploy, configure, and manage your Diffusion solution.

In this section:

- Installing
- Configuration
- Server
- Web server
- System management
- Network security
- Tuning
- Diagnostics
- Introspector
- Demos
- Tools

Installing

You can install the Diffusion server from a JAR file or through Red Hat Package Manager.

Review the system requirements before installing Diffusion.

Download Diffusion from the following location: http://download.pushtechnology.com/releases/5.5

The Diffusion installation includes a developer license that allows up to five concurrent connections to the Diffusion server. To use Diffusion in production, you can obtain a production license from Sales at Push Technology.

System requirements for the Diffusion server

Review this information before installing the Diffusion server.

The Diffusion server is certified on the system specifications listed here. In addition, the Diffusion server is supported on a further range of systems.

Certification

Push Technology classes a system as certified if the Diffusion server is fully functionally tested on that system.

We recommend that you use certified hardware, virtual machines, operating systems, and other software when setting up your Diffusion servers.

Support

In addition, Push Technology supports other systems that have not been certified.

Other hardware and virtualized systems are supported, but the performance of these systems can vary.

More recent versions of software and operating systems than those we certify are supported.

However, Push Technology can agree to support Diffusion on other systems. For more information, contact Push Technology.

Physical system

The Diffusion server is certified the following physical system specification:

- Intel Xeon E-Series Processors
- 8Gb RAM
- 8 CPUs
- 10 Gigabit NIC

NIC, CPU, and RAM (in decreasing order of importance) are the components that have the biggest impact on performance. High performance file system and disk are required. Intel hardware is used because of its ubiquity in the marketplace and proven reliability.

Virtualized system

The Diffusion server certified on the following virtualized system specification:

Host

- Intel Xeon E-Series Processors
- 32Gb RAM
- VMware vSphere 5.5

Virtual machine

- 8 VCPUs
- 8Gb RAM

Operating system

Diffusion is certified on the following operating systems:

- Red Hat 6.5, 6.6, and 7.1
- Windows Server 2012 R2

We recommend you install your Diffusion server on a Linux-based operating system with enterprise-level support available, such as Red Hat Enterprise Linux.

Java

The Diffusion server is certified on Oracle Java 8 64-bit JDK

Only the Oracle JDK is certified.

The Diffusion server also runs on Java Development Kit 7. However, Oracle withdrew support for Java 7 in April 2015. We recommend that you use the latest update of Java 8.

Ensure that you use the Oracle JDK and not the JRE.

Networking

Push Technology recommends the following network configurations:

- 10 Gigabit network
- F5 load balancers with SSL offloading

Client requirements

For information about the supported client platforms, see Interfaces supported and Supported client platforms.

Obtaining a Diffusion license

Diffusion includes a development license that enables you to use make up to 5 concurrent connections to the Diffusion server.

To use Diffusion in production, contact Sales at Push Technology for production licenses.

You can install a license file into an existing Diffusion server installation by placing the license file in the *`diffusion_installation`*`/etc` directory and restarting the server.

Installing the Diffusion server

The Diffusion binary files are available from the Push Technology website. You can install Diffusion using the graphical installer.

Before you begin

You must have Java installed on your system to install and use Diffusion.

About this task

To install Diffusion using the graphical installer, complete the following steps:

Procedure

1. Go to the Diffusion download page:
   http://download.pushtechnology.com/releases/5.5
2. Click on the following download links to download the required jar files into a temporary directory:
   - Diffusion (`Diffusion version_id.jar`)
   - Installer (`install.jar`)
3. In the temporary directory, double-click the `install.jar` file.
   The graphical installer launches.
4. Optional: If you have a production license, you can load it into the Diffusion installation at this point.
   You can skip this step if you are using the included development license.
   a) Ensure that the license file is available on your system.
   b) At the **Introduction** step, select **File** > **Load license file**
   c) In the window that opens, navigate to the license file (`licence.lic`). Click **Open**.
5. At the **Introduction** step, click **Continue**.
6. At the **License agreement** step, select **Accept** to accept the End User License Agreement (EULA) and click **Continue**.
7. At the **Destination directory** step, select the install destination.
   We recommend you create a `Diffusion` directory on your system. Click **Continue**.
8. At the **Select products** step, select the components you want to install.
   We recommend you select **All**. Click **Continue**.
9. At the **Confirmation** step, review the install information. If the information is correct, click **Continue** to confirm.
   The installer installs Diffusion into the directory specified.
10. At the **Summary** step, click **Done** to exit the graphical installer.

Results

You have successfully downloaded and installed Diffusion.

Related Links

Installed files on page 567

After installing Diffusion the following directory structure exists:

Installing Diffusion using the headless installer

The Diffusion binary files are available from the Push Technology website. You can install Diffusion from the command line.

Before you begin

You must have Java installed on your system to install and use Diffusion.

About this task

You can install in headless mode in circumstances where the graphical installer cannot be used or is not appropriate.

Procedure

1. Go to the Diffusion download page:
   http://download.pushtechnology.com/releases/5.5
2. Click on the following download links to download the required jar files into a temporary directory:
   - Diffusion (`Diffusion version_id.jar`)
   - Installer (`install.jar`)
3. Copy these files to a temporary directory on the system where Diffusion is to be installed.
4. In the terminal window, change to the directory where the Diffusion jar files are located.
5. Type the following command:

   ```
   java -jar install.jar Diffusionn.n.n.jar
   ```

   where *n.n.n* is the Diffusion release number.
6. If you agree to the terms of the license agreement, type `y` and Enter.
7. Enter the full path to the directory in which to install Diffusion and type Enter.
8. Type Y to install all packages.
   If you choose not to install all packages, the installer asks you about each package individually.

Results

Diffusion is installed in the specified directory.

What to do next

Your Diffusion installation includes a development license that allows connections from up to five clients. To use Diffusion in production, you can obtain a production license from Sales at Push Technology.

Copy the license file into the `/etc` directory of your Diffusion installation and restart Diffusion.

Related Links

Installed files on page 567

After installing Diffusion the following directory structure exists:

Installing Diffusion using Red Hat Package Manager

Diffusion is available as an RPM file from the Push Technology website.

About this task

On Linux systems that have Red Hat Package Manager installed, you can use it to install Diffusion.

Procedure

1. Go to the Diffusion download page:
   http://download.pushtechnology.com/releases/5.5
2. Click on the following download link to download the required RPM file:
   - Diffusion RPM (`Diffusion_n.n.n.rpm`)
3. Copy this file to a temporary directory on the system where Diffusion is to be installed.
4. In the terminal window, change to the directory where the Diffusion RPM file is located.
5. Type the following command:

   ```
   rpm -ivh  Diffusion_n.n.n.rpm
   ```

   where *n.n.n* is the Diffusion release number.

Results

Diffusion is installed in the following directory: `/opt/Diffusion`. A startup script is installed in the `/etc/init.d` directory that enables Diffusion to start when you start the system.

What to do next

Your Diffusion installation includes a development license that allows connections from up to five clients. To use Diffusion in production, you can obtain a production license from Sales at Push Technology.

Copy the license file into the `/etc` directory of your Diffusion installation.

Related Links

Installed files on page 567
After installing Diffusion the following directory structure exists:

Installing Diffusion using Docker

Diffusion is available as a Docker® image from Docker Hub.

Before you begin

You must have Docker installed on your system to run Diffusion from a Docker image. For more information, see https://docs.docker.com/userguide/ .

About this task

You can use Docker to install the Diffusion server, and a minimal complete set of its dependencies, on a Linux system. This image contains a Diffusion server with a trial license and default configuration and security.

Using Docker enables you to install the Diffusion server in an isolated and reproducible way.

Procedure

1. Pull the latest version of the Diffusion image.

```
docker pull pushtechnology/diffusion:latest
```

2. Run the image.

```
docker run -p 8080:8080 image_id
```

Where *image_id* is the ID of the image to run. Port 8080 is the port that is configured to allow client connections by default.

Results

Diffusion is now running in a container on your system. Clients can connect through port 8080.

Note: This Diffusion instance contains well known security principals and credentials. Do not use it in production without changing these values.

Next steps with Docker

The Diffusion image on Docker Hub includes the default configuration, default security, and trial license. Additional steps are required to secure and configure the Diffusion server.

Procedure

1. Create a Dockerfile that contains commands to configure a Diffusion image for your use.
   a) Base your Docker image on the Diffusion image.

```
FROM pushtechnology/diffusion:latest
```

   b) To use Diffusion in production, obtain a production license from Sales at Push Technology.

      The default Diffusion image includes a development license that allows connections from up to five clients.

   c) Copy the production license into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD license_file /opt/Diffusion/etc/licence.lic
```

      Where *license_file* is the path to the production license relative to the location of the Dockerfile.

   d) Create versions of the Diffusion configuration files that define your required configuration.

      For more information, see Using the configuration API on page 576.

   e) Copy these configuration files into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD configuration_file /opt/Diffusion/etc/file_name
```

Where *configuration_file* is the path to the configuration file relative to the location of the Dockerfile and *file_name* is the name of the configuration file.

f) Create versions of the `Security.store` and `SystemAuthentication.store` that define roles, principals and authentication actions for your security configuration.

For more information, see Pre-defined roles on page 141, DSL syntax: security store on page 336, and DSL syntax: system authentication store on page 323.

You can instead choose to edit these files using a Diffusion client. However, your Diffusion server is not secure for production use until you do so.

g) Copy these store files into the `/opt/Diffusion/etc` directory of your Diffusion image.

```
ADD store_file /opt/Diffusion/etc/file_name
```

Where *store_file* is the path to the store file relative to the location of the Dockerfile and *file_name* is the name of the store file.

h) Include any additional configuration actions you want to perform on your image in the Dockerfile.

2. Build your image.

Run the following command in the directory where your Dockerfile is located:

```
docker build .
```

Results

The image you created contains a configured Diffusion server ready for you to use in your solution. You can run multiple identically configured Diffusion servers from this image.

Updating your license file

You can update your Diffusion license file without having to restart the Diffusion server. Copy the new file over the old and ensure that the timestamp is updated.

Before you begin

Obtain a new or renewed license file from Push Technology.

About this task

When your license file expires, the Diffusion server continues to run for another day before it stops. We recommend you update your license file before your existing license file expires.

Procedure

1. Copy the new license file (`licence.lic`) over the existing file in the *`diffusion_directory`*`/etc` directory.

   You do not need to stop or restart the server.

2. Check that the timestamp of `licence.lic` has updated.

   - On Windows, you might have to use the following command to copy the file over and force the timestamp to update: `COPY /B licence.lic +,,`

3. Diffusion checks the timestamp of the `licence.lic` every minute. If the license file has been updated, Diffusion reloads it and logs this to stdout.

4. You can verify that the license file has been updated in the server by accessing the mbean **com.pushtechnology.diffusion** > **Server** > **LicenseExpiryDate**

Installed files

After installing Diffusion the following directory structure exists:

Table 75: Installed files

Folder name	Contents
`bin`	Executables for starting Diffusion
`clients`	Client Diffusion API libraries and related artifacts for all supported platforms.
`data`	Files used by publishers, the console, and third-party components. This directory is always on the server classpath. However, the `ext` directory is the preferred place to store resource files that are loaded by publishers.
`demos`	The compiled DAR files and source code for the demos issued with Diffusion. For more information, see Demos on page 782.
`deploy`	Publisher DAR files that are deployed when the Diffusion server starts. If you selected during the install process to deploy the demos, the demo DAR files are in this directory.
`docs`	License information, release notes, and install notes.
`etc`	Diffusion configuration files and example policy files for Silverlight and Flash. For more information, see Configuration on page 572.
`examples`	Example code that uses the Diffusion APIs.
`ext`	This directory, together with any jar files in this directory or subdirectories, are available through the classloader used to deploy application code to the Diffusion server. You can add library jar files to this directory that are required by application code such as publishers and local authentication handlers.
`html`	Files that are used by the default web server for issuables accessible through the browser.
`lib`	The main Diffusion server JAR file, third-party libraries, and additional server-side components.
`logs`	The directory to which Diffusion server and web server logs are written.
`stresstest`	The stress test package. For more information, see Stress test tuning on page 556.

Folder name	Contents
`tools`	Tools and utilities that help with testing and deploying Diffusion. For more information, see Tools and utilities on page 568
`xsd`	The schema files for the XML configuration files used by the server.

Tools and utilities

The following table describes the some of the contents of the `tools` directory.

> **Note:** The files present and their suffixes vary according to the platform that the product is installed on.

Table 76: Tools and utilities

Tool	Description
`/ec2`	A sample configuration for setting up the Diffusion server in an Amazon™ EC2 instance.
`/init.d`	Sample `init.d` files to start the Diffusion server as daemon on OS X®, Linux, or UNIX systems.
`/joyent`	A sample configuration for setting up the Diffusion server in a Joyent™ instance.
`DiffusionExternalClientTest.exe`	Windows external client test tool.
`extclient.bat/sh` and `tools.jar`	Generic external client test tool.
`externalclienttest.properties`	External client test tool properties.
`war.xml`	Example `war.xml` file
`web.xml` and `sun-web.xml`	Example `web.xml` files

Verifying your installation

Verifying the Diffusion installation

About this task

After installation, all of the Diffusion files are available in the directory specified during installation.

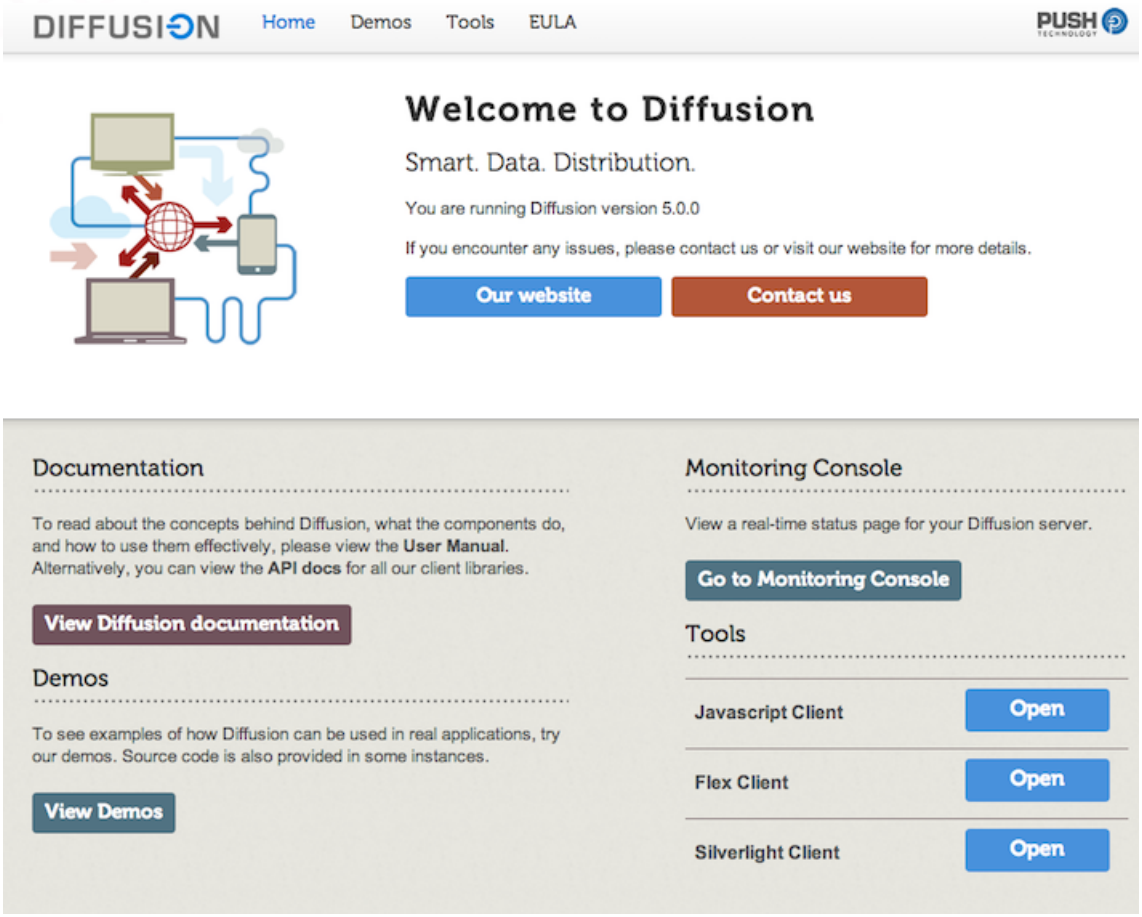
Procedure

1. Start the Diffusion server using one of the start script located in the `bin` directory of your Diffusion installation.
   - On Windows, use the `diffusion.bat` file.
   - On Linux, OS X, or UNIX, use the `diffusion.sh` file.
2. Inspect the log messages to ensure that the Diffusion server started successfully.

The terminal window displays logging information about the status of the Diffusion server. A log message containing the following text indicates that the server started successfully: `INFO|main|PUSH0165|Diffusion Server started.|com.pushtechnology.diffusion.DiffusionController` This line is typically the last one to be printed on terminal.

3. Inspect all log messages displayed in the terminal to search for `WARN` messages to ensure that all components have started correctly.
4. Open a browser and navigate to `http://serverAddress:8080` (or `http://localhost:8080`)

   The browser shows the Diffusion landing page.


DIFFUSION
Home
Demos
Tools
EULA
PUSH
Welcome to Diffusion
Smart. Data. Distribution.
You are running Diffusion version 5.0.0
If you encounter any issues, please contact us or visit our website for more details.
Our website
Contact us
Documentation
To read about the concepts behind Diffusion, what the components do, and how to use them effectively, please view the User Manual. Alternatively, you can view the API docs for all our client libraries.
View Diffusion documentation
Demos
To see examples of how Diffusion can be used in real applications, try our demos. Source code is also provided in some instances.
View Demos
Monitoring Console
View a real-time status page for your Diffusion server.
Go to Monitoring Console
Tools
Javascript Client
Open
Flex Client
Open
Silverlight Client
Open

   The landing page provides pointers to sources of information regarding legal terms and conditions (for example, EULA), user guides, API documentation and demos.

   The Diffusion server is ready to be used.

5. If you chose to install the demos, you can access them from the landing page. Use these demo publishers to verify your installation.

Web server installation

Diffusion can act as a web server by modifying the `etc/Connectors.xml` and adding a web-server definition to a connector.

About this task

This explains how to setup a servlet for Diffusion. You might find it easier to create and deploy a web application using the Ant script discussed below. Add the following lines to `WEB-INF/web.xml`. The `servlet` section needs to go with the other `servlet` sections. A `servlet-mapping` entry is not required for the `DiffusionServlet` as it does not handle any requests.

```
<servlet>
    <servlet-name>Diffusion</servlet-name>
    <display-name>Diffusion Servlet</display-name>
    <servlet-
class>com.pushtechnology.diffusion.servlet.DiffusionServlet</
servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Make sure that the JAR files in `ext` and the `lib/diffusion.jar` are in the lib directory of `WEB-INF`. You should also copy the `lib/thirdparty` directory and its contents to `WEB-INF/lib`. Also copy all of the properties files from the `etc` directory to the `classes` directory. Make sure that the policy files have the correct path in the `etc/Connectors.xml` file. The `html/lib/DIFFUSION` directory must also be copied into the `webapp` directory as `lib/DIFFUSION`, this directory contains the browser API libraries.

Creating a web application archive

In the `tools` directory of the default installation is an Ant build file `war.xml` that will package the Diffusion into a WAR file that you can deploy to a servlet container. Invoking the `war` or `all` target of the file builds the archive at `build/diffusion.war`. The diffusion.war file can be deployed to your servlet container according to its documentation.

A WAR file is an archive like a JAR that contains a WEB-INF directory. This directory contains XML based configuration describing the web application and the directories `classes` and `lib` that contain `.class` or `.jar` files that are added to the classpath. The top level of the WAR file contains resources that can be served by Tomcat™. Place the configuration files for Diffusion in the `WEB-INF/classes` directory. Place the `diffusion.jar` and files from the `ext` directory and the `thirdparty` directory itself in the `WEB-INF/lib` directory.

Tomcat Configuration

Diffusion uses the `java.util.prefs` functionality so Tomcat must be started with the userRoot configured. Without this warning messages are generated in the logs. The Tomcat user requires write permissions to the userRoot directory.

```
-Djava.util.prefs.userRoot=/var/lib/tomcat6/diffusion/prefs/user
```

Tomcat must have connectors defined to handle incoming connections. A connector defines the port, protocol, and various properties of how the connection is handled. Connectors are defined in the `Server.xml` file. See the Tomcat documentation for more information. The following is an example connector for handling HTTP 1.1 connections on port 8080.

```
<Connector port="8080"
        connectionTimeout="20000"
        URIEncoding="UTF-8"
```

```
        maxThreads="3"
        protocol="HTTP/1.1" />
```

Accessing publishers from Tomcat

Diffusion started within Tomcat allows Tomcat to access the publishers. Tomcat can be used to serve JSP files providing dynamically generated content. These files can access publishers using the publishers class static methods.

```
<%@ page
 import="java.util.List,com.pushtechnology.api.publisher.*" %>
<html>
    <head>
        <title>Publisher Information</title>
    </head>
    <body>
        <table>
            <tr>
                <th>Publisher Name</th>
                <th>Topics</th>
            </tr>
            <% for (Publisher pub : Publishers.getPublishers())
 { %>
            <tr>
                <td><%= pub.getPublisherName() %></td>
                <td><%= pub.getNumberOfTopics() %></td>
            </tr>
            <% } %>
        </table>
    </body>
</html>
```

The above is the content of a JSP file that return a list of the publisher Diffusion is running with the number of topics each publisher owns.

Apache Mod Proxy installation

Apache Mod Proxy can be used to forward HTTP requests from an Apache web server to Diffusion. It does not support persistent connections orWebSockets so the WebSocket and HTTPC connections do not work. Make sure that you include the following into the Apache configuration file (Virtual host setting).

```
ProxyPass /diffusion/ http://localhost:8080/diffusion/
```

For more information, see the Apache Mod Proxy documentation.

Apache AJP13 Installation

Apache AJP can be used to forward requests from an Apache web server to Tomcat. In the Apache virtual host configuration, mount the path

```
JkMount /diffusion/*dfnjetty
```

Workers definition file

```
worker.dfnjetty.port=8009
worker.dfnjetty.host=(host IP)
worker.dfnjetty.type=ajp13
worker.dfnjetty.lbfactor=1
worker.dfnjetty.cachesize=50
worker.dfnjetty.socket_keepalive=1

worker.list=dfnjetty
```

A connector that handles the AJP/1.3 protocol is needed running on port 8009 (because of the Workers file described above). See the Tomcat documentation for more information on this.

IIS Installation

Use an ISAPI_Rewrite tool. For example, http://www.helicontech.com/isapi_rewrite

The rewrite rule is as follows:

```
RewriteEngine on RewriteRule ^diffusion/ http://localhost:8080/
diffusion/ [p]
```

Diffusion home directory

The servlet container must be aware of the Diffusion. Add the path to the directory that contains the Diffusion JAR file to the Java VM arguments that you use to start the servlet container.

`-Ddiffusion.home=`*`diffusion_installation`*`/lib`

Licensing

The Diffusion license must be loaded by the servlet container. Add the path to the license file to the JVM arguments that you use to start the servlet container.

`-Ddiffusion.license.file=`*`diffusion_installation`*`/etc/licence.lic`

The license file is usually located in the `etc` directory of the Diffusion installation. As this file must be referenced when the container starts, it is not a part of the Diffusion servlet. The path must be absolute or relative to the servlet container, not the web application.

Keystore

You must tell the servlet container which keystore to use. Add the path to the keystore to the JVM arguments that you use to start the servlet container.

`-Ddiffusion.keystore.file=`*`diffusion_installation`*`/etc/`*`keystore_name`*

The keystore file is usually located in the `etc` directory of the Diffusion installation. As this file must be referenced when the container starts, it is not a part of the Diffusion servlet. The path must be absolute or relative to the servlet container, not the web application.

Diffusion configuration

The built-in Diffusion web server is configured using the `WebServer.xml` file. When using a third party web server at least some of this functionality can be disabled. The file-service, and two http-service entries can be removed as Tomcat provides this functionality. The client-service is needed to supportWebSocket, HTTPC and HTTP connection protocols. If these are not used it can be disabled as well.

Configuration

You can configure the Diffusion server using XML files which normally reside in the `etc` folder.

Alternatively, a Diffusion server can be instantiated in a Java application and configured programmatically. Some properties can also be changed at runtime programmatically from within publishers.

In a Java client environment certain properties can also be configured programmatically.

All properties (whether configured from XML or programmatically) are available to read programmatically from within the Java API.

XML configuration

Configuring a Diffusion server using XML property files

XML Property files

A Diffusion server is configured using a set of XML property files typically loaded from the `etc` folder. In a new Diffusion installation example versions of these files are provided which can be edited as required.

XML is used rather than standard property files due to the hierarchic nature and the ability to support repeating groups.

The Introspector has a built in configuration editor, which is able to load and save the configuration files remotely if required.

XSD files are issued that define the content of the XML property files and this section summarizes the XSD content.

Configuration path loading

You can pass a parameter to Diffusion upon startup so that files are not automatically loaded from the `etc` folder but loaded from a different folder. This folder does not have to contain the complete set of XML files, but the file is loaded from the specified folder first, if it exists. If it does not, Diffusion loads the configuration file from the `etc` folder. When Diffusion starts, it logs where each configuration file has been loaded from.

XML Value types

When XML values are loaded, the schema is checked so that we know that it is valid, but to aid configuration, there are some extra data types. When values are loaded, they are trimmed of leading and trailing white space.

Table 77: XML Value types

Data type	Meaning
push:boolean	true or false
push:string	String value
push:int	A number between -2,147,483,648 and 2,147,483,647
push:long	A number between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807
push:double	A number between 2-1074and (2-2-52)¬†.21023
push:port	A positive number but less than 65535
push:millis	A string that represents the number of milliseconds. Append the mnemonic for the time unit. The mnemonic can be either upper or lower case. **s** Seconds **m** Minutes **h**

Data type	Meaning
	Hours **d** Days 360000, 360s, 6m all represent 6 minutes
push:bytes	A string that represents the number of bytes. Append the mnemonic size unit. The mnemonic can be either upper or lower case. **k** Kilobytes **m** Megabytes **g** Gigabytes 6291456, 6144k, 6m, all represent 6 Megabytes
push:log-level	A log level can be ERROR, WARN, INFO, DEBUG, or TRACE.
push:percent	A value that represents a percentage, this can have the trailing percent sign (%)
push:positiveNonZeroInt	A number between 1 and 2,147,483,647
push:positiveInt	A number between 0 and 2,147,483,647
push:positiveNonZeroLong	A number between 1 and 9,223,372,036,854,775,807
push:positiveLong	A number between 0 and 9,223,372,036,854,775,807
push:threadPriority	A number between 1 and 10
<element>	This notation is used to indicate a complex element type. It can also be List<element> to indicate a repeating property group.

Environmental values

When defining custom configurations, you can define environmental variables that can be reused in all XML property files. These variables can be defined in the `etc/Env.xml` property file to be used in all other property files. Suppose, for example, the `etc/Env.xml` file defines a `server-name` variable, with value `d-unit` as follows:

```
<env>
 <property name="server-name">d-unit</property>
</env>
```

The `server-name` variable can be used in all other property files, where the value `d-unit` is appropriate, either as a value for an attribute, as in

```
<server name={server-name}>,Ä¶</server>
```

or as a name for an element as in:

```
<server>{server-name}</server>
```

As a side remark, it is worth noting that names can be combined to provide malleable environmental variables. Suppose for instance `Env.xml` contains the following entries:

```
<env>
 <property name="server-name">myServer</property>
 <property name ="server-version">V2.0</property>
</env>
```

Then `server-name` and `server-version` can be combined, for instance within the same `etc/Env.xml`, as

```
<property name="server-and-version">{server-name}-{server-
version}</property>
```

and used in all other configuration files.

Obfuscated values

Obfuscation is a technique through which sensitive entries can be hidden in clear text. Within the Diffusion context, obfuscation can be used to hide password, and any other data deemed to be sensitive, to be included in configuration files.

To create obfuscated values you can use the **Property Obfuscator** dialog in the Diffusion plugin for Eclipse™. Obfuscated entries are identified by a `OB:` prefix in clear text.

Related Links

Property obfuscator on page 781
This dialog is part of the Diffusion perspective and can be used to hide sensitive Diffusion configuration file entries, such as passwords and JMS login credentials.

Property obfuscator on page 781
This dialog is part of the Diffusion perspective and can be used to hide sensitive Diffusion configuration file entries, such as passwords and JMS login credentials.

Programmatic configuration

An alternative to configuring a Diffusion server using XML property files is to instantiate a Diffusion server within a Java application and configure it programmatically before starting it.

Using this technique, you can do without XML property files altogether as every aspect of the XML configuration can also be supplied programmatically.

If desired, some properties can be loaded from XML files and some supplied programmatically or default properties can be bootstrapped from XML files and overridden programmatically before the server is started.

Most server properties can be configured only before the server is started. Instantiate the server within an application and configure before starting the server. However, certain configuration items (examples being conflation and connection policies) can be configured at any time during the life of the server. The API documentation makes it clear if a property can be changed at runtime.

Because the properties that can be set programmatically reflect those that can be set in XML this section does not describe the properties in detail. The XSD property descriptions or the API documentation for the configuration API can be consulted for full details.

As well as allowing configuration properties to be set the configuration API also allows all properties that can be configured to be read at runtime. So publisher code has direct access to all property settings.

Using the configuration API

General use

From within a Java application the root of the configuration tree can be obtained at any time using `ConfigManager.getConfig()`. This provides access to the general objects and can be used from within server-side or client-side code.

From within server-side code (for example, a publisher) the server configuration root can be obtained using `ConfigManager.getServerConfig()` which exposes all of the server side configuration also.

From the configuration root you can navigate to any subordinate configuration objects to view them or set their properties.

On the server side most properties cannot be changed after the server has started and they become locked so any attempt to change them results in an exception. Certain properties (such as conflation and connection policies) can be changed at runtime. The API documentation makes it clear which properties can be changed at runtime.

In client-side Java code the configuration does not become locked and can be changed at any time. However, some values are read at the start only. Ideally, set all properties before creating any client side objects.

For configuration objects which are optional but there can be many (multiplicity 0..n), there are appropriate add methods to add new objects. For example to add a publisher and set a property on it:

```
PublisherConfig publisher =
    ConfigManager.getServerConfig().addPublisher(
        "MyPublisher",
        "com.pub.MyPublisher");
publisher.setTopicAliasing(false);
```

In these cases there are also methods to obtain the full list (for example, `getPublishers()`) or to obtain a specific one by name (for example, `getPublisher("MyPublisher")`). In many cases there are also methods to remove an object.

> **Note:** When there must be at least one object (multiplicity 1..n), you must configure at least one. However, if a server is started with missing configuration of this kind, suitable defaults are normally created and a warning logged.

Single instance configuration objects (multiplicity 1..1) subordinate to the root can be obtained so that their properties can be changed (or read). So, for example the `Queues` object (an instance of `QueuesConfig`) can be obtained using the `getQueues()` method.

When a single configuration object is optional (multiplicity 0..1), the `get` method can return null if it has not been defined. In this case to set it the set method (as opposed to add) returns the object created. An example of this is the file service (`FileServiceConfig`) on a web server (`WebServerConfig`) as shown in the following example code:

```
ServerConfig config = ConfigManager.getServerConfig();
WebServerConfig webServer = config.addWebServer("MyWebServer");
FileServiceConfig fileService = webServer.setFileService("File
 Service");
```

Configuring a server

After instantiating a Diffusion server in Java the root of the server configuration tree can be obtained from the server object itself and configuration objects can be navigated to and changed as required before starting the server.

For example, the following code shows how to add a connector that accepts client connections on port 9090:

```
DiffusionServer server = new DiffusionServer();
ServerConfig config = server.getConfig();
ConnectorConfig connector = config.addConnector("Client
 Connector");
connector.setPort(9090);
connector.setType(Type.CLIENT);
server.start();
```

In reality, it is best to configure far more values. However, if any essential objects are omitted (such as queues), suitable defaults are created when the server starts and a warning is logged.

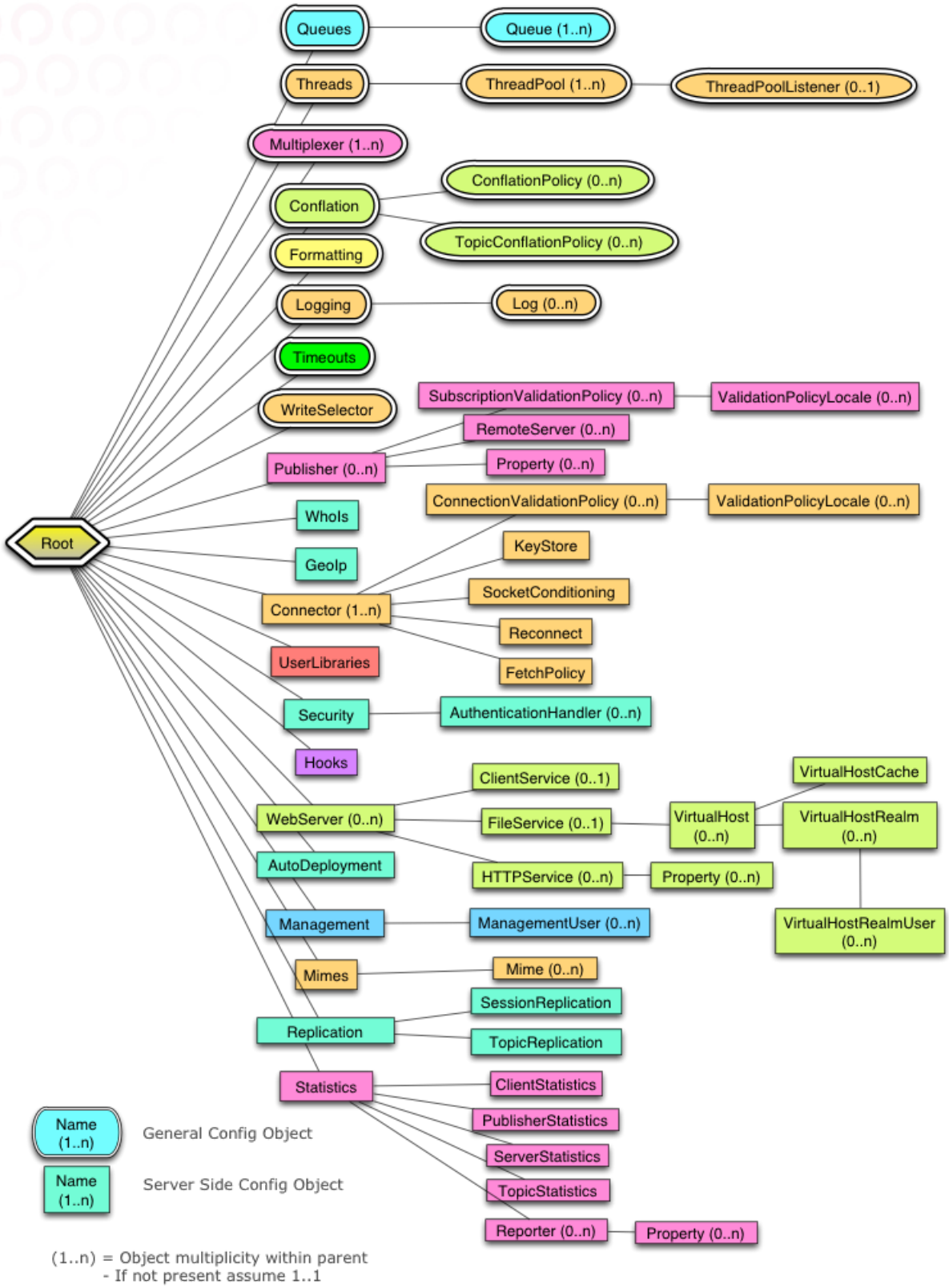
Configuration access from a publisher

Within a publisher the configuration object for the publisher itself can be obtained using the `getConfig` method which returns the publisher configuration (`PublisherConfig`) object.

Related Links

The configuration tree on page 578

The configuration tree



All general objects can be obtained by navigating from the root object obtained from `ConfigManager.getConfig()`.

Server-side objects can be reached only in a server environment using `ConfigManager.getServerConfig()`.

Related Links

Using the configuration API on page 576

Configuring replication

You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Procedure

1. Edit the `Replication.xml` file to configure replication.

```
<replication enabled="true">
    <provider>HAZELCAST</provider>
    <sessionReplication enabled="true" />
    <topicReplication enabled="true">
        <topics>
            <topicPath>foo/bar</topicPath>
        </topics>
    </topicReplication>
</replication>
```

   - In the `replication` element, set `enabled` to `true` to enable replication.
   - In the `sessionReplication` element, set `enabled` to `true` to configure the server to reflect client session information into the datagrid.
   - In the `topicReplication` element, set `enabled` to `true` to configure the server to reflect topic information and topic data into the datagrid.
   - Inside the `topics` element, use one or more `topicPath` elements to define the topics to which to apply topic replication and failover of the active update source.

     The content of the `topicPath` element is a path to a single topic. Topic replication and failover of the active update source are applied to the topic defined by the path and all topics below it in the topic tree.

     Unlike a topic selector, the topic path does not contain any leading or trailing characters. For example, use `<topicPath>foo/bar</topicPath>` to select the topic `foo/bar`.

     The `topicPath` elements also define which sections of the topic tree are have update sources created for them by the server.
2. Restart the Diffusion server to load the configuration.
3. Ensure that your clients are configured to reconnect if they lose their connection to the server.

Related Links

Replication.xml on page 580

This file specifies the schema for the replication properties.

Replication.xml

This file specifies the schema for the replication properties.

replication

Properties defining replication.

The following table lists the attributes that an element of type `replication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether replication is enabled for this server.	true

The following table lists the elements that an element of type `replication` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
provider	push:string	The type of replication provider to use to replicate the data. Currently only Hazelcast is supported.	1	1
sessionReplication	sessionReplication	The definition for session replication	1	1
topicReplication	topicReplication	The definition for topic replication	1	1

sessionReplication

Properties defining session replication.

The following table lists the attributes that an element of type `sessionReplication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether session replication is enabled for this server.	true

topicReplication

Properties defining topic replication.

The following table lists the attributes that an element of type `topicReplication` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether topic replication is enabled for this server.	true

The following table lists the elements that an element of type `topicReplication` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topics	topics	The topics that are configured to use replication.	1	1

topics

Properties defining the topics to replicate.

The following table lists the elements that an element of type `topics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topicPath	push:string	A topic path that identifies the root of a tree that will be replicated by this server.	0	unbounded

Related Links

Session replication on page 83
You can use session replication to ensure that if a Classic API client connection fails over from one server to another the state of the client session is maintained.

Topic replication on page 86
You can use topic replication to ensure that the structure of the topic tree, topic definitions, and topic data are synchronized between servers.

Failover of active update sources on page 87
You can use failover of active update sources to ensure that when a server that is the active update source for a section of the topic tree becomes unavailable, another server is assigned to be the active update source for that section of the topic tree. Failover of active update sources is enabled for any sections of the topic tree that have topic replication enabled.

Configuring replication on page 579
You can configure replication by editing the `etc/Replication.xml` files of your Diffusion servers.

Configuring your datagrid provider on page 581
You can configure your datagrid to work correctly with your solution architecture.

Configuring your datagrid provider

You can configure your datagrid to work correctly with your solution architecture.

Configuring Hazelcast

By default, the Hazelcast node in your Diffusion server multicasts to all other Hazelcast nodes in your network.

To define which Hazelcast nodes can communicate with each other, use the `hazelcast.xml` configuration file. For example, you can configure the Hazelcast nodes in the development environment to communicate only with other nodes in the development environment and not with Hazelcast nodes in the production environment.

The following example shows the structure of the `hazelcast.xml` file:

```
<hazelcast xsi:schemaLocation="http://www.hazelcast.com/schema/
config
```

```
    http://www.hazelcast.com/schema/config/hazelcast-
config-3.0.xsd"
    xmlns="http://www.hazelcast.com/schema/config"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <properties>
        <property name="hazelcast.logging.type">slf4j</property>
    </properties>

    <network>

          <join>
            <!-- <multicast enabled="true" /> -->
            <tcp-ip enabled="true">
                <members>
                    <member>node1.example.com</member>
                    <member>203.0.113.1</member>
                    <member>203.0.113.2:5757</member>
                    <member>203.0.113.3-7</member>
                </members>
            </tcp-ip>
        </join>

    </network>
</hazelcast>
```

This example configuration disables the multicast capability and defines the Hazelcast nodes that can be connected to.

The Hazelcast nodes can be defined by hostname, by IP address, or by IP range. The default port used by Hazelcast is 5701. If you want to connect on a different port, you can specify this when you define the node, using the format *host*:*port*.

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the *`diffusion_installation`*`/data` directory. Restart the Diffusion server to load the configuration.

For more information about using the `hazelcast.xml` file to configure Hazelcast, see the Hazelcast™ Reference Manual.

Diagnosing problems with Hazelcast

If you enable logging for Hazelcast, you can use the log files to diagnose problems with Hazelcast.

To enable logging, include the following line in your `hazelcast.xml` file:

```
<property name="hazelcast.logging.type">slf4j</property>
```

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the *`diffusion_installation`*`/data` directory. Restart the Diffusion server to load the configuration.

You can also enable logging by starting the Diffusion server that contains the node with the following parameter `-Dhazelcast.logging.type=slf4j`

You can enable JMX for your Hazelcast nodes and use a JMX tool to examine the MBeans.

To enable JMX for a Hazelcast node, include the following line in your `hazelcast.xml` file:

```
<property name="hazelcast.jmx">true</property>
```

Ensure that the `hazelcast.xml` file is on the Diffusion server classpath. For example, by putting the file in the *`diffusion_installation`*`/data` directory. Restart the Diffusion server to load the configuration.

You can also enable JMX by starting the Diffusion server that contains the node with the following parameter `-Dhazelcast.jmx=true`

For more information about using Hazelcast, see the Hazelcast™ Reference Manual.

Related Links

Replication.xml on page 580
This file specifies the schema for the replication properties.

Configuring the JMS adapter

Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Use the `Publishers.xml` file to define and deploy the JMS adapter as a publisher on your Diffusion server:

```
<publisher name="JMSAdapter">
  <class>com.pushtechnology.diffusion.adapters.jms.JMSAdapter</
class>
  <enabled>true</enabled>
  <start>true</start>
</publisher>
```

Related Links

JMS on page 104
Consider whether to incorporate JMS providers into your solution.

Transforming JMS messages into Diffusion messages or updates on page 106
JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

Sending messages using the JMS adapter on page 110
The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

Publishing using the JMS adapter on page 109
The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

Using JMS request-response services with the JMS adapter on page 112
You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

Example: Configuring JMS providers for the JMS adapter on page 592
Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

Example: Configuring topics for use with the JMS adapter on page 593
Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

Example: Configuring messaging with the JMS adapter on page 595
Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the

`JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

Example: Configuring pubsub with the JMS adapter on page 594
Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

Example: Configuring the JMS adapter to work with JMS services on page 596
Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

JMSAdapter.xml

This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

JMSRootConfig

The mandatory root node of the JMS adapter configuration.

The following table lists the elements that an element of type `JMSRootConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topics	JMSTopicsConfig	The set of Diffusion topics created at startup.	0	1
providers	JMSProvidersConfig	The set of JMS providers.	1	1

JMSTopicsConfig

The set of Diffusion topics created at startup. Diffusion Unified client messaging does not require an existing topic, but Diffusion publishing and Diffusion Classic client messaging do.

The following table lists the elements that an element of type `JMSTopicsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
stateful	JMSStatefulTopicConfig	The configuration required to create a stateful Diffusion topic, including the initial state of the topic.	0	unbounded
stateless	JMSTopicConfig	The configuration required to create a Diffusion topic.	0	unbounded

JMSTopicConfig

The configuration required to create a Diffusion topic.

The following table lists the attributes that an element of type `JMSTopicConfig` can have:

Name	Type	Description	Required
name	xs:string	The full topic path of the topic to create. For example, 'foo/bar/baz'.	true

Name	Type	Description	Required
reference	xs:string	A string value to associate with this topic. For example, a description of the topic.	false

JMSStatefulTopicConfig

The following table lists the attributes that an element of type `JMSStatefulTopicConfig` can have:

Name	Type	Description	Required
initialState	xs:string	The initial state of a topic when the topic is created.	false

JNDIPropertiesConfig

The set of named values required to to create an InitialContext to access the JNDI configuration of the JMS server. Individual JMS providers will provide documentation on this step.

The following table lists the elements that an element of type `JNDIPropertiesConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
property	JNDIProperty	A named value.	0	unbounded

JNDIProperty

A named value.

The following table lists the attributes that an element of type `JNDIProperty` can have:

Name	Type	Description	Required
name	xs:string	The property name.	true
value	xs:string	The property value.	true

JMSProviderConfig

The configuration model to connect to a JMS provider (a broker), establish sessions, subscribe, and publish to destinations.

The following table lists the attributes that an element of type `JMSProviderConfig` can have:

Name	Type	Description	Required
name	xs:string	The name associated with this configuration model.	false

The following table lists the elements that an element of type `JMSProviderConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
jndiProperties	JNDIPropertiesConfig	The set of named values required to to create an InitialContext to access the JNDI configuration of the JMS server.	1	1
jmsProperties	JMSConnectionConfig	The configuration related to connection to the JMS provider.	1	1
sessions	JMSSessionsConfig	The configuration for all JMS sessions related to this JMS provider.	1	1
reconnection	JMSReconnectionConfig	The configuration for reconnection behavior.	0	1
subscriptions	JMSSubscriptionsConfig	The set of subscriptions to JMS destinations.	0	1
publications	JMSPublicationsConfig	The set of publications to JMS destinations.	0	1

JMSProvidersConfig

The set of JMS providers.

The following table lists the elements that an element of type `JMSProvidersConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
provider	JMSProviderConfig	The configuration model to connect to a JMS provider.	0	unbounded

JMSSessionsConfig

The configuration for all JMS sessions related to this JMS provider.

The following table lists the elements that an element of type `JMSSessionsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
anonymousSessions	JMSAnonymousSessionsConfig	A number of JMS sessions shared between JMSSubscriptions.	1	1
namedSessions	JMSNamedSessionsConfig	The set of named JMS sessions, optionally used by JMSSubscription nodes in order to guarantee ordering, or use specific JMS session properties.	0	1

JMSAnonymousSessionsConfig

A number of JMS sessions shared between JMSSubscriptions.

The following table lists the attributes that an element of type `JMSAnonymousSessionsConfig` can have:

Name	Type	Description	Required
number	PositiveInteger	The number of shared JMS sessions	true

JMSNamedSessionsConfig

The set of named JMS sessions, optionally used by JMSSubscription nodes in order to guarantee ordering, or use specific JMS session properties.

The following table lists the elements that an element of type `JMSNamedSessionsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
session	JMSNamedSessionConfig	A named set of configuration relating to the placing of a JMS session.	1	unbounded

JMSReconnectionConfig

Following a disconnection event the adapter optionally attempts periodic reconnection. The first reconnection attempt occurs after minFrequency seconds, and the following after twice that number. The back-off time value doubles until it reaches the maxFrequency value in seconds. For example, where minFrequency=2 and maxFrequency=10, the reconnection will be attempted after 2s, 4s, 8s, 10s, 10s and so on.

The following table lists the attributes that an element of type `JMSReconnectionConfig` can have:

Name	Type	Description	Required
minFrequency	PositiveInteger	The interval between disconnection and the first reconnection attempt (in seconds). The interval is doubled for each subsequent reconnection attempt.	true
maxFrequency	PositiveInteger	The maximum interval between reconnection attempts.	true

JMSSubscriptionsConfig

The set of subscriptions to JMS destinations.

The following table lists the elements that an element of type `JMSSubscriptionsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
subscription	JMSSubscriptionConfig	Configuration to subscribe to a JMS destination and relay to Diffusion topics or messaging or both.	0	unbounded

JMSPublicationsConfig

The set of publications to JMS destinations.

The following table lists the elements that an element of type `JMSPublicationsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
publication	JMSPublicationConfig	Configuration to receive Diffusion topic messaging and relay to a JMS destination.	0	unbounded

JMSSubscriptionConfig

Configuration to subscribe to a JMS destination and relay to Diffusion topics

The following table lists the attributes that an element of type `JMSSubscriptionConfig` can have:

Name	Type	Description	Required
sessionName	xs:string	The name of the session to use. This session name must be defined in the namedSessions element. If this element is not defined, the JMS adapter does not start.	false

The following table lists the elements that an element of type `JMSSubscriptionConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
destination	JmsURI	The URI of the JMS destination.	1	1
options	JMSSubscriptionOptions	Configuration relating to publishing in JMS.	0	1
messaging	ClientMessagingConfig	Configuration relating to the sending of a Diffusion message to a single Diffusion client.	0	1
publish	TopicPublishingConfig	Configuration relating to the publishing of a message or setting of a topic's state.	0	1

JMSPublicationConfig

Configuration to receive Diffusion topic messaging and relay to a JMS destination.

The following table lists the elements that an element of type `JMSPublicationConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
destination	JmsURI	The URI of the JMS destination.	1	1
options	JMSPublicationOptionsConfig	Configuration relating to publishing to JMS destinations.	0	1
messaging	ClientMessagingConfig	Configuration relating to the sending of a Diffusion message to a single Diffusion client.	0	1

JMSSubscriptionOptions

Options employed when subscribing to a JMS destination.

The following table lists the attributes that an element of type `JMSSubscriptionOptions` can have:

Name	Type	Description	Required
noLocal	xs:boolean	Inhibits the delivery of messages published through its own connection.	false

The following table lists the elements that an element of type `JMSSubscriptionOptions` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
selector	xs:string	SQL 92 compliant expression used to filter messages received from a JMS destination.	0	1

JMSPublicationOptionsConfig

The following table lists the attributes that an element of type `JMSPublicationOptionsConfig` can have:

Name	Type	Description	Required
ttl	PositiveInteger	The Time-To-Live value for a published JMS message, in milliseconds	false
priority	JMSPriorityRange	The higher the number, the higher the priority.	false
deliveryMode	JMSDeliveryMode	Maps to javax.jms.DeliveryMode	false

ClientEndpointConfig

The following table lists the attributes that an element of type `ClientEndpointConfig` can have:

Name	Type	Description	Required
topicName	xs:string	The topic path used by this end point. Depending on the task it might not need to relate to an existing topic.	true

The following table lists the elements that an element of type `ClientEndpointConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
transformation	MessageTransformationConfig	The transformation type to use for messages relayed to and from this topic.	0	1

MessageTransformationConfig

The following table lists the attributes that an element of type `MessageTransformationConfig` can have:

Name	Type	Description	Required
type	MessageTransformationType	The transformation employed when relaying Diffusion to JMS messages, or JMS to Diffusion messages.	false

The following table lists the elements that an element of type `MessageTransformationConfig` can contain:

Name	Type	Description	Minimum occurrences	Maximum occurences
sessionProperties	SessionPropertyMappings	The set of session property mappings.	0	1

TopicPublishingConfig

Configuration relating to the publishing of a message or setting of a topic's state.

ClientMessagingConfig

Configuration relating to the sending of a Diffusion message to a single Diffusion client.

The following table lists the attributes that an element of type `ClientMessagingConfig` can have:

Name	Type	Description	Required
routingProperty	xs:string	The routingProperty attribute describes a facet of the JMS TextMessage that contains the destination Diffusion client SessionID.	false

SessionPropertyMapping

A mapping from Diffusion session properties to JMS message metadata (JMS headers or properties).

The following table lists the attributes that an element of type `SessionPropertyMapping` can have:

Name	Type	Description	Required
from	xs:string	Currently limited to $Principal	true
to	xs:string	Values starting with "JMS" are mapped into JMS headers (for example, JMSType), others are mapped into JMS message properties.	true

SessionPropertyMappings

The set of SessionPropertyMappings.

The following table lists the elements that an element of type `SessionPropertyMappings` can contain:

Name	Type	Description	Minimum occurrences	Maximum occurences
sessionProperty	SessionPropertyMapping	A session property name. Currently, only $Principal is supported.	1	1

JMSCredentialsConfig

A username and password pair.

The following table lists the elements that an element of type `JMSCredentialsConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
username	xs:string	A username to use to connect to the JMS provider.	1	1
password	xs:string	The password associated with the username.	1	1

JMSConnectionConfig

The configuration related to connection to the JMS provider.

The following table lists the attributes that an element of type `JMSConnectionConfig` can have:

Name	Type	Description	Required
connectionFactoryName	xs:string	The name of the connection factory to use.	true

The following table lists the elements that an element of type `JMSConnectionConfig` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
credentials	JMSCredentialsConfig	Optional credentials, used when connecting to the JMS provider	0	1

JMSSessionConfig

All configuration relating to the placing of a JMS session.

The following table lists the attributes that an element of type `JMSSessionConfig` can have:

Name	Type	Description	Required
transacted	xs:boolean	Currently unsupported.	false
acknowledgeMode	JMSSessionAcknowledgeMode	Currently unsupported.	false

JMSNamedSessionConfig

A JMSSessionConfig that can be referred to by name.

The following table lists the attributes that an element of type `JMSNamedSessionConfig` can have:

Name	Type	Description	Required
name	xs:string	Name used to refer to the session elsewhere in the JMSProviderConfig.	true

Related Links

JMS on page 104
Consider whether to incorporate JMS providers into your solution.

Transforming JMS messages into Diffusion messages or updates on page 106
JMS messages are more complex than Diffusion content. A transformation is required between the two formats.

Sending messages using the JMS adapter on page 110
The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

Publishing using the JMS adapter on page 109
The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

Using JMS request-response services with the JMS adapter on page 112
You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

Example: Configuring JMS providers for the JMS adapter on page 592
Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

Example: Configuring topics for use with the JMS adapter on page 593
Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

Example: Configuring messaging with the JMS adapter on page 595
Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

Example: Configuring pubsub with the JMS adapter on page 594
Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

Example: Configuring the JMS adapter to work with JMS services on page 596
Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

Example: Configuring JMS providers for the JMS adapter

Use the `providers` element of the `JMSAdapter.xml` configuration file to define the JMS providers that the JMS adapter can connect to.

ActiveMQ

You can connect to an ActiveMQ instance by defining a `provider` element that contains the required JNDI, credentials, and session information. See the following example:

```
<providers>
    <provider name="myActiveMQ">
        <jndiProperties>
            <property name="java.naming.factory.initial"
 value="org.apache.activemq.jndi.ActiveMQInitialContextFactory"/
>
            <property name="java.naming.provider.url"
 value="tcp://hostname:61616"/>
        </jndiProperties>

        <jmsProperties
 connectionFactoryName="ConnectionFactory">
```

```
            <credentials>
                <username>user</username>
                <password>password</password>
            </credentials>
        </jmsProperties>

        <sessions>
            <anonymousSessions number="1" transacted="false"
 acknowledgeMode="AUTO_ACKNOWLEDGE" />
        </sessions>

    </provider>

</providers>
```

IBM MQ

You can connect to an IBM MQ instance by defining a `provider` element that contains the required information. See the following example:

```
<providers>
    <provider name="myIBMMQ">
        <jndiProperties>
            <property name="java.naming.factory.initial"
 value="com.sun.jndi.fscontext.RefFSContextFactory"/>
            <property name="java.naming.provider.url"
 value="hostname:1414"/>
        </jndiProperties>

        <jmsProperties connectionFactoryName="CF2"
            <sessions>
                <anonymousSessions number="2" transacted="false"
 acknowledgeMode="AUTO_ACKNOWLEDGE"/>
            </sessions>
        </jmsProperties>

    </provider>
</providers>
```

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring topics for use with the JMS adapter

Use the `topics` element of the `JMSAdapter.xml` configuration file to define the Diffusion topics that the JMS adapter uses. These topics are created when the JMS adapter starts.

The following example shows the definitions for a stateless and a stateful topic:

```
<topics>
    <stateless name="example/updates/stateless"/>
```

```
    <stateful name="example/updates/stateful"
 initialState="rhubarb"/>
</topics>
```

- The JMS adapter cannot create or use topics that are in a branch of the topic tree that is created by another publisher or client. For example, if example/updates already exists and was created by a Diffusion client, the JMS adapter cannot create and use example/updates/stateless.
- Similarly, other clients and publishers cannot create or use topics in a branch of the topic tree that was created by the JMS adapter.
- When defining a topic to create, all intermediate topics on the topic path are also created. For example, when the JMS adapter creates example/updates/stateless it also creates example and example/updates as stateless topics. All intermediate topics are created as stateless topics.
- All stateful topics are created as single value topics.
- When defining a stateful topic, you must set the initial state of the topic.

Related Links

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring pubsub with the JMS adapter

Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define JMS adapter subscriptions to JMS destinations and the Diffusion topics to publish updates to.

The following example shows subscriptions to JMS destinations defined by the `destination` elements. When the JMS adapter receives an update message through the subscription, it publishes that update message to the Diffusion topic defined in the corresponding `publish` element.

```
<subscriptions>
    <subscription>
        <destination>jms:topic:EXAMPLE.UPDATE.TOPIC</
destination>
        <publish topicName="example/updates/stateless"/>
    </subscription>
    <subscription>
        <destination>jms:topic:EXAMPLE.UPDATE.TOPICTWO</
destination>
        <publish topicName="example/updates/stateful"/>
    </subscription>
</subscriptions>
```

The Diffusion topics must be defined in the `topics` section of the `JMSAdapter.xml` configuration file.

Related Links

Publishing using the JMS adapter on page 109

The JMS adapter can publish data from a JMS destination onto topics in the Diffusion topic tree.

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring messaging with the JMS adapter

Use the `publications` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients send messages to JMS destinations. Use the `subscriptions` element of the `JMSAdapter.xml` configuration file to define how Diffusion clients receive messages from JMS destinations.

From Diffusion clients to JMS destinations

The following example defines the Diffusion topic path through which the JMS adapter receives messages from a Diffusion client and the JMS destination to send those messages to.

```
<publications>
    <publication>
        <destination>jms:queue:EXAMPLE.REQUEST.QUEUE</
destination>
        <messaging
                  topicName="example/topic/requests"
                  routingProperty="JMSCorrelationID">
            <transformation type="basic">
                <sessionProperties>
                  <sessionProperty from="$Principal"
 to="diffusionPrincipal"/>
                </sessionProperties>
            </transformation>
        </messaging>
    </publication>
</publications>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.
- The `transformation` section defines how a message is transformed between a JMS message and a Diffusion message. For more information, see Transforming JMS messages into Diffusion messages or updates on page 106.
- The `sessionProperties` section defines whether the Diffusion session properties of the client that sends the message are included as JMS headers or properties in the transformed message. Currently, only $Principal is supported.

From JMS destinations to Diffusion clients

The following example defines the JMS destination that the JMS adapter retrieves messages on and the Diffusion topic path through which the JMS adapter relays those messages to a Diffusion client.

```
<subscriptions>
    <subscription>
        <destination>jms:queue:EXAMPLE.UPDATE.QUEUE</
destination>
```

```
        <options noLocal="true"/>
        <messaging
               topicName="example/direct/messages"
               routingProperty="JMSCorrelationID"/>
    </subscription>
</subscriptions>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.
- The `noLocal` attribute of the `options` element defines whether the JMS adapter does not retrieve messages from a JMS queue that it is the originator of.

Related Links

Sending messages using the JMS adapter on page 110
The JMS adapter can send messages from a Diffusion client to a JMS destination and messages from a JMS destination to a specific Diffusion client.

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

Example: Configuring the JMS adapter to work with JMS services

Use the `publications` and `subscriptions` elements of the `JMSAdapter.xml` configuration file to define the message flow for using Diffusion with JMS services.

In the following example, the `publications` section defines the JMS destination to put request messages on and the Diffusion topic path through which the JMS adapter receives those request messages from a Diffusion client. The `subscriptions` section defines the JMS destination that the JMS adapter retrieves response messages on and the Diffusion topic path through which the JMS adapter relays those messages to a Diffusion client.

```
<publications>
    <publication>
        <destination>jms:queue:REQUEST.QUEUE</destination>
        <messaging topicName="example/requests"
 routingProperty="JMSCorrelationID"/>
    </publication>
</publications>
<subscriptions>
    <subscription>
        <destination>jms:queue:RESPONSE.QUEUE</destination>
        <options noLocal="true">
            <selector>JMSCorrelationID like '${serverName}/%'</
selector>
        </options>
        <messaging topicName="example/responses"
 routingProperty="JMSCorrelationID"/>
    </subscription>
</subscriptions>
```

- The `routingProperty` attribute describes the JMS header or property that the JMS adapter uses to set or get the client session ID.

- The `noLocal` attribute of the `options` element defines whether the JMS adapter does not retrieve messages from a JMS queue that it is the originator of.
- When subscribing to a JMS destination, the JMS adapter can use selectors. In this example, the selector used requires that the routing property, in this case `JMSCorrelationID`, contains the name of the Diffusion server where the JMS adapter is deployed. This prevents the JMS adapter from consuming messages that are not intended for it.

  The JMS adapter replaces the variable `${serverName}` with the name of its server.

Related Links

Using JMS request-response services with the JMS adapter on page 112
You can use the messaging capabilities of the JMS adapter to interact with a JMS service through request-response.

Configuring the JMS adapter on page 583
Use the `JMSAdapter.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

JMSAdapter.xml on page 584
This file specifies the schema for the configuration required by the JMS Adapter. Note that JMS topics and queues are referred to only as destinations. Topics refers exclusively to Diffusion topics.

DEPRECATED: Configuring the legacy JMS adapter version 5.1

Use the `JMSAdapter51.xml` configuration file to configure the JMS adapter to send and receive messages with destinations on a JMS server.

Note: We recommend that you use the new JMS adapter. This legacy version is deprecated.

Use the `Publishers.xml` file to define and deploy the JMS adapter as a publisher on your Diffusion server:

```
<publisher name="JMSAdapter51">

 <class>com.pushtechnology.diffusion.adapters.jms.JMSAdapter51</
class>
  <enabled>true</enabled>
  <start>true</start>
  <property name="config.filename">../etc/JMSAdapter51.xml</
property>
</publisher>
```

Related Links

DEPRECATED: Legacy JMS adapter on page 113
For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Receiving data from JMS on page 115
Diffusion clients can receive data from a JMS provider through the JMS adapter. A client can receive updates from a JMS topic or messages from a JMS queue.

Sending messages to JMS on page 116
Diffusion clients can send messages to a JMS provider through the JMS adapter.

Processing a request-reply message with a Diffusion client on page 117

A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

Sending a request-reply message from a Diffusion client on page 118
You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

JMS adapter data flow examples on page 115
The examples in this section show how data flows between the Diffusion server and a JMS provider.

Configuring the JMS Adapter v5.1

The configuration file for the legacy JMS adapter v5.1 is typically called `JMSAdapter51.xml`, although you can override this by using a setting in `Publishers.xml`.

Procedure

1. Configure `Publishers.xml`

   Instantiate the JMS adapter by enabling it in `etc/Publishers.xml`, for example:

```
<publisher name="JMSAdapter">

 <class>com.pushtechnology.diffusion.adapters.jms.JMSAdapter</
class>
  <enabled>true</enabled>
  <start>true</start>
  <property name="config.filename">../etc/JMSAdapter.xml</
property>
</publisher>
```

2. Configure `JMSAdapter51.xml`

   A `JMSAdapter51.xml` for ActiveMQ can look like this:

```
<property name="use.global.session">false</property>
```

   Similarly, a sample `JMSAdapter51.xml` TIBCO Enterprise Message Service™ looks like this:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<jms-config>
  <binding>
    <env>
      <property name="java.naming.factory.initial">
        com.tibco.tibjms.naming.TibjmsInitialContextFactory
      </property>
      <property name="java.naming.provider.url">
        tcp://localhost:7222
      </property>
      <property name="java.naming.security.principal">
 jndi_username
      </property>
      <property name="java.naming.security.credentials">
 jndi_password
      </property>
    </env>
    <connection-factory name="ConnectionFactory">
      <credentials username="jms_username"
 password="jms_password"/>
```

```
      <reconnect>
 <max-reconnections>10</max-reconnections>
 <interval>5000</interval>
      </reconnect>
    </connection-factory>
    <root-topic>jms/tibco</root-topic>
    <priority low="3" high="7" />
    <queue-distribution-mode>SMALLEST_QUEUE</queue-
distribution-mode>
  </binding>
  <mapping>
    <artefact-names jms=".$" diffusion="/~"/>
  </mapping>
</jms-config>
```

TIBCO Enterprise Message Service requires that a `ConnectionFactory` definition is provided in `factories.conf`, for example:

```
[ConnectionFactory]
  type = generic
  url = tcp://localhost:7222
  ssl_verify_host = disable
```

This has been tested with TIBCO Enterprise Message Service 7.0; see the TIBCO Enterprise Message Service documentation for further details.

For IBM MQ v7.x, the binding section can look like this:

```
<env>
  <property
 name="java.naming.factory.initial">com.sun.jndi.fscontext.RefFSContextFa
property>
  <property name="java.naming.provider.url">file:///var/mqm/
jndi</property>
</env>
```

IBM MQ can include MQRFH2 headers in messages sent between JMS and non-JMS systems. For control over this behavior, set the property in `mq.target.client` in `Publishers.xml`; to disable the headers, set this value to `1`. For the default behavior, do not provide the property.

```
<property name="mq.target.client">1</property>
```

Some JMS vendors (for example, IBM MQ) require a JMS `Session` for each topic or queue subscription. The default configuration for the JMS adapter does not allow for this, but you can enable it by setting the `use.global.session` property to `false` in `Publishers.xml`:

```
<property name="use.global.session">false</property>
```

Table 78: Properties that can be specified when configuring the JMS adapter

<env>	All properties within the <env> tag of `JMSAdapter51.xml` are used when creating the `InitialContext` which is in turn used to create the connection to the JMS server.

<connection-factory>	This tag specifies the name of the connection factory to use. This varies between JMS vendors and the server configuration.
<credentials>	Optionally, specify a username and/or password which is used to create a JMS client connection. However, most JMS implementations are likely to have restricted access for anonymous clients or clients which do not require authentication so you can specify a user here who has the necessary privileges to receive and send messages to the JMS destinations that are exposed through Diffusion.
<reconnect>	If a connection cannot be made between Diffusion and the JMS server or the connection is severed, the `<max-reconnections>` and `<interval>` parameters enable you to specify how many times to retry the connection before giving up and how long to wait between each attempt. A value for `<max-reconnections>` of -1 indicates that the adapter keeps trying to connect forever.
<root-topic>	To provide partitioning of the topic tree between topics related to JMS and other topics, it is necessary for a root topic name is defined here. In the event of more than one JMS adapter, you can segment the topic tree further.
<priority>	JMS supports messages with up to 10 priority levels (0-9) with 0-4 considered to be different grades of normal priority and 5-9 to be different grades of high priority. Diffusion only has the concept of low, medium and high priority. Using this parameter, you can map JMS messages within a given priority range to a representative Diffusion priority, and in the other direction.
<queue-distribution-mode>	Unlike topics, a message on a JMS queue is delivered to only one client. When Diffusion receives a message from a queue, it uses this parameter to determine which of its connected clients subscribed to the corresponding Diffusion topic is selected to receive that message. Valid values are: **SMALLEST_QUEUE**

	Choose a client with the fewest number of messages outstanding in its message queue from Diffusion. **RANDOM** Select a client randomly.
<artefact-names>	Not all characters in a Diffusion topic name are valid JMS topic or queue names (and the other way around). The two attributes on this element (`jms` and `diffusion`) are lists of characters where the n^{th} character in one is replaced by the corresponding n^{th} character in the other.

Related Links

DEPRECATED: Legacy JMS adapter on page 113
For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Receiving data from JMS on page 115
Diffusion clients can receive data from a JMS provider through the JMS adapter. A client can receive updates from a JMS topic or messages from a JMS queue.

Sending messages to JMS on page 116
Diffusion clients can send messages to a JMS provider through the JMS adapter.

Processing a request-reply message with a Diffusion client on page 117
A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

Sending a request-reply message from a Diffusion client on page 118
You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

JMS adapter data flow examples on page 115
The examples in this section show how data flows between the Diffusion server and a JMS provider.

jms-config

The following table lists the elements that an element of type `jms-config` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
binding	binding		1	1
mapping	mapping		1	1

binding

The following table lists the elements that an element of type `binding` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
env	env		1	1
connection-factory	connection-factory	A collection of properties required to create an InitialContext to access the JNDI configuration of the JMS server.	1	1
root-topic	push:string	The JMS Adapter uses the topic name specified here as the top-level Diffusion topic, with all subtopics appearing as children beneath it.	1	1
priority	priority	Specify how JMS message priorties (in the range 0-9) map to Diffusion message priorities high, normal or low.	0	1
queue-distribution-mode	push:jmsQueueDistributionMode	Messages received from JMS queues are only sent to one Diffusion client. This element allows the distribution strategy to be chosen from RANDOM (choose any client) or SMALLEST_QUEUE (choose the client with the fewest pending messages in its queue from Diffusion).	1	1
topic-aliases	topic-aliases	This element is not currently used.	0	unbounded

mapping

The following table lists the elements that an element of type `mapping` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
artefact-names	artefact-names	Diffusion and JMS topic/queue names can both contain different reserved characters. The attributes on this element create a substitution table of characters. If a Diffusion topic name contains a character in the 'diffusion' attribute, it is replaced with the character in the same position of the 'jms' attribute when translating to a JMS topic or queue name (and vice-versa).	1	1
topic-map	jms-artefact-map	This element is not currently used.	0	unbounded
queue-map	jms-artefact-map	This element is not currently used.	0	unbounded

env

The following table lists the elements that an element of type `env` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
property	property	This element represents a property which is supplied when creating the JNDI object (InitialContext) from which JMS resources are obtained.	0	unbounded

connection-factory

Details for resolving the ConnectionFactory object from the nominated JNDI server.

The following table lists the attributes that an element of type `connection-factory` can have:

Name	Type	Description	Required
name	push:string		false

The following table lists the elements that an element of type `connection-factory` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
credentials	credentials	Username and password pair (optional), used when creating the JMS Connection from the ConnectionFactory.	0	1
reconnect	reconnect	This element is supplied to control the reconnection policy for the initial connection between Diffusion and the JMS server, and any subsequent reconnections that might be necessary.	0	1

jms-artefact-map

The following table lists the attributes that an element of type `jms-artefact-map` can have:

Name	Type	Description	Required
jms	push:string		true
descendants	push:boolean		false

artefact-names

The following table lists the attributes that an element of type `artefact-names` can have:

Name	Type	Description	Required
jms	push:string		true
diffusion	push:string		true

property

A named string property

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true

credentials

The following table lists the attributes that an element of type `credentials` can have:

Name	Type	Description	Required
username	push:string		false
password	push:string		false

priority

The following table lists the attributes that an element of type `priority` can have:

Name	Type	Description	Required
low	push:jmsMessagePriority		false
high	push:jmsMessagePriority		false

topic-aliases

The following table lists the elements that an element of type `topic-aliases` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topic	topic		0	unbounded

topic

The following table lists the attributes that an element of type `topic` can have:

Name	Type	Description	Required
name	push:string		true
alias	push:string		true

reconnect

The following table lists the elements that an element of type `reconnect` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
max-reconnections	push:int	When attempting a connection between Diffusion and the JMS server, this parameter specifies how many attempts can be made before giving up. A value of -1 (the default) means to retry forever.	1	1

Name	Type	Description	Minimu occurre	Maximum occurences
interval	push:millis	This parameter specifies how long to wait between reconnection attempts between Diffusion and the JMS server.	1	1

Related Links

DEPRECATED: Legacy JMS adapter on page 113
For backwards compatibility, Diffusion provides an older version of the JMS adapter, version 5.1. The version 5.1 JMS adapter is not compatible with the latest version of the JMS adapter.

Receiving data from JMS on page 115
Diffusion clients can receive data from a JMS provider through the JMS adapter. A client can receive updates from a JMS topic or messages from a JMS queue.

Sending messages to JMS on page 116
Diffusion clients can send messages to a JMS provider through the JMS adapter.

Processing a request-reply message with a Diffusion client on page 117
A common pattern among JMS solutions is for a client to receive a message from JMS and a reply is expected to be sent to a specific JMS topic or queue.

Sending a request-reply message from a Diffusion client on page 118
You can send a message from a Diffusion client into a JMS server with the expectation that a JMS client processes the message and sends a response back to the same Diffusion client.

JMS adapter data flow examples on page 115
The examples in this section show how data flows between the Diffusion server and a JMS provider.

Aliases.xml

This file specifies the schema for the aliases properties used in a web server.

aliases

List of aliases

The following table lists the elements that an element of type `aliases` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
alias	alias	An alias definition	0	unbounded

alias

An alias definition

The following table lists the attributes that an element of type `alias` can have:

Name	Type	Description	Required
name	push:string	A name for the alias.	true

The following table lists the elements that an element of type `alias` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
source	push:string	The source URL, which can be expressed as a regular expression.	1	1
destination	push:string	The destination path.	1	1

ConnectionValidationPolicy.xml

This file specifies the schema for the connection validation policy.

connection-validation-policies

Connection validation policies

The following table lists the elements that an element of type `connection-validation-policies` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
policy	policy	A connection validation policy.	0	unbounded

policy

A connection validation policy.

The following table lists the attributes that an element of type `policy` can have:

Name	Type	Description	Required
name	push:string	Each policy must be supplied with a unique name for easy reference.	true
type	push:string	The policy type should be either "blacklist" or "whitelist". A blacklist indicates that if any of the policy rules in this policy match the incoming connection, that connection is to be rejected. A whitelist requires that at least one policy rule matches for the connection to be accepted.	true
automatic	push:boolean	Policies which are set to automatic are applied by Diffusion and the publishers do not need to perform any checks themselves. If this attribute is set to false, the policy is not applied unless it is done so by the publisher. If a value is not specified, a default of true is used.	false

The following table lists the elements that an element of type `policy` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
addresses	addresses	Connection validation policy addresses. These are addresses that are blacklisted/ whitelisted.	0	1
locale	locale	Connection validation policy locale. This is locale details that are blacklisted/ whitelisted.	0	unbounded

addresses

The following table lists the elements that an element of type `addresses` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
address	push:string	An IP address (or regular expression) of a connecting client.	0	unbounded
hostname	push:string	The hostname (or regular expression) of a connecting client.	0	unbounded
resolved-name	push:string	The resolved hostname (or regular expression) of a connecting client, as returned by the WhoIs service.	0	unbounded

locale

The following table lists the elements that an element of type `locale` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
country	push:string	The ISO country code of the connecting client, as returned by the WhoIs service.	0	1
language	push:string	The ISO language code of the connecting client, as returned by the WhoIs service.	0	1

Connectors.xml

This file specifies the schema for the connectors properties.

connectors

Connectors

The following table lists the elements that an element of type `connectors` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
connector	connector	Connector definition	0	unbounded

connector

Connector definition

The following table lists the attributes that an element of type `connector` can have:

Name	Type	Description	Required
name	push:string	The connector name	true

The following table lists the elements that an element of type `connector` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
type	connectorType	The type of connection supported. By default 'all' types are supported but the connector can be restricted to one of the following specific types - 'client' (Clients only), or 'policy' (Policy File Requests only).	0	1
api-type	connectorApiType	This setting constrains the API that can be used with this connector. The allowed values are 'all', 'classic', and 'unified'. The value 'unified' indicates that clients must use the Unified API. The value 'classic' indicates that clients must use the Classic API. The value 'all' indicates that clients can use either API. The default value is 'classic'.	0	1
host	push:string	The name or the IP address that the connector binds to. This is optional.	0	1
port	push:port	The port on which the connector accepts connections.	1	1
acceptors	push:positiveNonZeroInt	DEPRECATED: The number of acceptors to run for this connector. If this value is not specified, a default of 1 is used.	0	1
backlog	push:positiveNonZeroInt	The maximum queue length for incoming clients. If a connection indication arrives when the queue is full, the connection is refused. If a value is not specified, a default of 1000 is used.	0	1
socket-conditioning	socket-conditioning	Describes the properties associated with TCP socket connections.	1	1
web-server	push:string	If this connector is required to serve HTTP requests, this element specifies a web-server entry in WebServer.xml. If a value is not specified, the connector cannot serve HTTP requests.	0	1
policy-file	push:string	The location/name of the policy file if this connector is required to act as a policy file server (type='all' or 'policy').	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
validation-policy-file	push:string	The location/name of a connection validation policy file to use for this connector. Applies only to type 'all' or 'client'.	0	1
key-store	key-store-definition	Keystore details for any connector that is to support secure (SSL) connections. If this is not specified, SSL connections are not supported.	0	1
queue-definition	push:string	An optional queue definition to use for this connector. This applies only to connectors of type 'all' or 'client'. The definition must exist in Server.xml. If this is not specified, the default queue definition specified in Server.xml is used.	0	1
reconnect	reconnect	Optional reconnection properties which apply only to connectors that accept 'client' connections. If this is not specified, reconnection of client connections is not supported.	0	1
ignore-errors-from	ignore-errors-from	Specifies addresses from which connection errors can be ignored. This is useful for masking errors that might be reported due to the connector port being pinged by some known external entity.	0	1
thread-pool-definition	push:string	Optionally, this can be used to specify a thread pool definition to be used for this connector to create its own inbound thread pool. If specified, the thread pool definition must exist in Server.xml. If a value is not specified, the default inbound thread pool is used.	0	1
selector-thread-pool-definition	push:string	Optionally, this can be used to specify a selector thread pool definition to be used for this connector to deal with NIO operations. If specified, the selector thread pool definition must exist in Server.xml. If a value is not specified, the default selector thread pool is used.	0	1
system-ping-frequency	push:millis	This indicates the interval at which clients are pinged by the server to ensure that they are still connected. If a response is not received from the client before the expiry of another interval period, the client is assumed to be disconnected. If this is not specified or a value of 0 is supplied, clients are not automatically pinged.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
fetch-policy	fetch-policy	Specifies a policy for batching fetch requests. If a value is not specified, no policy is applied and fetches are not batched.	0	1
proxy-protocol	proxyProtocol	Indicates the proxy protocol required for connection. Can have the values 'NONE' or 'HA_PROXY'. The default value is 'NONE'. Only connections with the protocol specified are allowed. A non-'NONE' value should only be allowed on connectors that are not publicly accessible. 'HA_PROXY' reffers to the Proxy protocol that was first implemented by HAProxy but it is also supported by others including Amazon's Elastic Load Balancer.	0	1

socket-conditioning

Describes properties associated with TCP socket connections.

The following table lists the elements that an element of type `socket-conditioning` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
input-buffer-size	push:bytes	Specifies the size of the socket input buffer to use for each connection. If a value is not specified, a default of 1M is used. The greater of this value and the max-message-size set in Server.xml is used when setting the socket input buffer size.	0	1
output-buffer-size	push:bytes	This value specifies the size of the output buffer to use for each connection. This must be large enough to accomodate the largest message to be sent. Messages are 'batched' into this buffer and so the larger the buffer, the more messages can be sent in a single write. If a value is not specified, a default of 64k is used.	0	1
keep-alive	push:boolean	This enables or disables TCP keep-alive. If a value is not specified, a default of true is used.	0	1
no-delay	push:boolean	This enables or disables TCP_NODELAY (disable/enable Nagle's algorithm). If a value is not specified, a default of true is used.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
reuse-address	push:boolean	When a TCP connection is closed the connection can remain in a timeout state for a period of time after the connection is closed (typically known as the TIME_WAIT state or 2MSL wait state). For applications using a well-known socket address or port, it might not be possible to bind a socket to the required SocketAddress if there is a connection in the timeout state involving the socket address or port. Enabling this feature allows the socket to be bound even though a previous connection is in a timeout state. If this value is not specified, the feature is enabled.	0	1

reconnect

Reconnect properties.

The following table lists the elements that an element of type `reconnect` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
keep-alive	push:millis	This specifies the period within which a disconnected client can reconnect to the same client session. Messages for the client continue to be queued during this period.	1	1
max-depth	push:positiveInt	As messages continue to be queued for a client whilst it is disconnected, this enables you to specify a larger maximum queue size that is used during the period that the client is disconnected. When the client reconnects, the maximum reverts back to its previous size (once any backlog had been cleared). If the specified size is not greater than the current maximum size, this has no effect. If this value is not specified, a default of 0 is used which means that no attempt is made to extend the queue size when a client is disconnected.	0	1

key-store-definition

The keystore definition that allows SSL connection to a connector.

The following table lists the attributes that an element of type `key-store-definition` can have:

Name	Type	Description	Required
mandatory	push:boolean	If this is set to true, all connections must use this keystore and SSL connection is mandatory. If a value is not specified, a default of false is used, meaning that the connector accepts either SSL or non-SSL connections.	false

The following table lists the elements that an element of type `key-store-definition` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
file	push:string	The keystore file path.	1	1
password	push:string	The password for the keystore.	1	1

ignore-errors-from

Some external monitors cause the Diffusion server to log errors, as it is not a valid Diffusion connection. Adding the remote IP address to this list ensure that the errors are not logged.

The following table lists the elements that an element of type `ignore-errors-from` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
ip-address	push:string	An IP address or unknown if the remote IP address is being masked.	1	unbounded

fetch-policy

This is the policy for batching fetch requests. This can be used when fetches on topic sets might be large and lead to an excessive number of fetch reply messages being queued for a client at one time. The policy can define that the replies are sent in periodic batches to allow the client time to process them and prevent client queues filling.

The following table lists the elements that an element of type `fetch-policy` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
batch-size	push:positiveInt	Specifies the maximum number of fetch reply messages to send per batch. If this is set to 0, no batching occurs.	1	1
delay	push:millis	Specifies the time period between submissions of batches. If a batch size is specified, this must be a positive value.	1	1

Related Links

Connectors on page 655

An introduction to the concept of connectors within a Diffusion server.

Env.xml

This file specifies the schema for the environment properties.

env

The following table lists the elements that an element of type `env` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
property	property	Environment variable value	0	unbounded

property

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	xsd:token	Name of the environment variable.	true

Logs.xml

This file specifies the schema for the log properties.

logs

Properties defining logging options.

The following table lists the elements that an element of type `logs` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
console-log-level	push:log-level	The log level to start console logging at. Can be ERROR, WARN, INFO, DEBUG, or TRACE. If a value is not specified, a default of INFO is used.	0	1
log-message-data	push:boolean	Indicates whether the data part of messages is logged as part of routine diagnostic message logging (FINE and FINEST levels). If this is false, credentials message headers are also hidden. If a value is not specified, a default of true is used.	0	1
server-log	push:string	The log to use for the server. This must specify the name of a configured log definition.	1	1

Name	Type	Description	Minimu occurre	Maximum occurences
default-log-directory	push:string	The default log folder for all logs, although this can be over-ridden for each log.	1	1
async-logging	push:boolean	Indicates whether logging is asynchronous. Asynchronous logging is performed by a separate thread as opposed to being performed in-line by the logging thread. This is normally set to true for performance reasons, but asynchronous logging might cause problems in some OS environments. This element provides the option to turn asynchronous logging off, if so advised. If a value is not specified, a default of true is used.	0	1
logging-queue-size	push:positiveInt	The size of the asynchronous logging queue. In normal cases, leave this value at the default value of 128k entries.	0	1
thread-name-logging	push:boolean	Indicates whether the thread name is logged with messages. If this is not specified, thread names are logged.	0	1
log	log	A log definition.	0	unbounded

log

A log definition.

The following table lists the attributes that an element of type `log` can have:

Name	Type	Description	Required
name		Name of the log definition	true
rotation-period	push:positiveNonZeroInt	A time period that the log exists for before being rotated. This is a positive non-zero integer, with unit specified by rotation-unit.	false
rotation-unit	push:timeunit	A time unit to specify the unit used alongside rotation-period. This can be "day(s)", "hour(s)", "minute(s)".	false

The following table lists the elements that an element of type `log` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
log-directory	push:string	The name of the directory to which this log file is written. If a value of not specified, the default-log-directory is used.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
file-pattern	push:string	This is used to specify the name of the log file. The following values can be used within the pattern. "/" - the local pathname separator. "%t" - the system temporary directory. "%g" - the generation number to distinguish rotated logs. "%h" - the value of the "user.home" system property. "%u" - a unique number to resolve conflicts. "%s" - the system type - for example, 'Diffusion'. "%n" - the system name as defined in Server.xml. "%d" - the date as specified in diffusion.properties (date.format), this is included when using daily rotation. "%%" - translates to a single percent sign "%". If a log file name is not specified, a default of "%s.log" is used.	0	1
level	push:log-level	Specifies the starting log level. This can be ERROR, WARN, INFO, DEBUG, or TRACE. If a value is not specified, a default of INFO is used.	0	1
xml-format	push:boolean	Indicates whether the log file is output in XML format. If a value is not specified, a default of false is used.	0	1
date-format	push:string	Specifies a date format to name a log. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd" is used.	0	1
file-limit	push:bytes	Specifies an approximate maximum amount to write (in bytes) to any one log file. If this is zero, there is no limit. If a value is not specified, a default of 0 is used.	0	1
file-append	push:boolean	Specifies whether log records are appended to existing log files. If a value is not specified, a default of false is used and log files are overwritten.	0	1
file-count	push:positiveNonZeroInt	Specifies the number of log files to use. Must be at least 1. If a value is not specified, a default of 1 is used.	0	1
rotate-daily	push:boolean	Indicates whether the log is to rotate on a daily basis. This is superceded by rotation-period.	0	1

Management.xml

This file specifies the schema for the management properties that enable JMX access over an RMI JMXConnectorServer.

management

The management configuration.

The following table lists the attributes that an element of type `management` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies if an RMI JMXConnectorServer is enabled, making JMX remotely available.	true

The following table lists the elements that an element of type `management` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
host	push:string	The local interface used for the RMI registry and the JMX service.	0	1
registry-port	push:port	The RMI registry port. If a value is not specified, a default of 1099 is used.	0	1
connection-port	push:port	The JMX service port. If a value is not specified, a default of 1100 is used.	0	1
assigned-roles	assigned-roles	Additional roles granted to principals that authenticate successfully. Used to grant all JMX users a base set of permissions.	0	1
register-topics	push:boolean	Vestigial setting with zero effect. Superseded by features in Statistics.xml.	0	1
users	users	The management users. Registering JMX-only users here is deprecated. Instead configure appropriate users in the system authentication store, or with a custom authentication handler, and assign the users the role ADMINISTRATOR or VIEW_SERVER based upon whether they need full access to JMX operations or only to monitor the server.	1	1

assigned-roles

Assigned security roles.

The following table lists the elements that an element of type `assigned-roles` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
role	push:string	A security role.	1	unbounded

users

Management users.

The following table lists the elements that an element of type `users` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
user	user	A user that can use the JMX interface.	1	unbounded

user

Management user.

The following table lists the elements that an element of type `user` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
name	push:string	User name for JMX credentials.	1	1
password	push:string	Password for JMX credentials.	1	1
read-only	push:boolean	Specify if user has read-only access. If this value is false, the user is assigned their standard roles, the roles configured for assigned-roles, and the JMX_ADMINISTRATOR role. By default, JMX_ADMINISTRATOR is configured to provide full administrator access. Otherwise, the user is assigned only their standard roles and the assigned-roles, which may not grant sufficient permissions to perform operations that affect the state of the server.	1	1

Mime.xml

This file specifies the schema for the mime properties.

mimes

The following table lists the elements that an element of type `mimes` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
mime	mime	Mime.	0	unbounded

mime

The following table lists the attributes that an element of type `mime` can have:

Name	Type	Description	Required
type	push:string	Mime type.	true
extension	push:string	Mime extension.	true

Publishers.xml

This file specifies the schema for the publisher properties.

publishers

The set of publishers that the Diffusion server is aware of at startup.

The following table lists the elements that an element of type `publishers` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
publisher	publisher	A publisher definition.	0	unbounded

publisher

A publisher definition.

The following table lists the attributes that an element of type `publisher` can have:

Name	Type	Description	Required
name	push:string	The publisher name.	true

The following table lists the elements that an element of type `publisher` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topics	push:string	An optional, comma-separated list of topic names specifying topics to be automatically created for the publisher as it is started. This technique does not allow for topics to be set up with data and so it is more usual to define the topics you require in the initialLoad method of the Publisher. This property remains mostly for backwards compatibility.	0	1
class	push:string	The full class name of a Java class that implements the publisher. This class must extend the Java API Publisher class and provide implementations of methods as required. The class file must be available on the classpath of the Diffusion server (or in the configured usr-lib or ext folder).	1	1
enabled	push:boolean	By default, the publisher is loaded as the server starts. By setting this to false, the publisher is not loaded.	0	1
start	push:boolean	By default, the publisher is started after it is loaded. By specifying this as false, the publisher can be loaded but not started and then can be started later using JMX.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
topic-aliasing	push:boolean	Specifies whether topic aliasing is turned on for all topics created by the publisher. If the value is true, a short topic alias is transmitted in delta messages instead of the full topic name. By default, this is true, but because there are certain limitations when using topic aliasing there might be situations where you might want to turn it off.	0	1
ack-timeout	push:millis	This specifies the default ACK (message acknowledgment) timeout value (in milliseconds) to use for messages sent from the publisher that require acknowledgment and do not have a timeout explicitly specified. If a value is not specified, a default of 1s is used.	0	1
auto-ack	push:boolean	Indicates whether to automatically acknowledge messages sent from clients to the publisher requiring acknowledgment. By default, this is false so messages requiring acknowledgment must be manually acknowledged by the publisher.	0	1
subscription-policy-file	push:string	Path of a subscription validation policy file. If this value is specified, the file is used to validate client subscriptions to topics owned by the publisher.	0	1
stop-server-if-not-loaded	push:boolean	If this is set to true and the publisher fails to load, the Diffusion server stops. By default, this is false.	0	1
log-level	push:log-level	Specifies the log level for the publisher. If this value is not specified, the publisher logs at the default log level.	0	1
server	server	A specification of a server that is automatically connected to by the publisher when it starts.	0	unbounded
web-server	web-server	If the publisher has associated web content, it can be deployed with the publisher by specifying this property.	0	1
launch	launch	Launch detail describes how the publisher might be accessed externally, if it has an associated webpage.	0	unbounded
property	property	A property available to the publisher. This can be used to configure publisher-specific variables or parameters.	0	unbounded

launch

Launch detail.

The following table lists the attributes that an element of type `launch` can have:

Name	Type	Description	Required
name	push:string	The launcher name.	true
category	push:string	An optional category to which this launcher belongs. For example, "demo" for the Diffusion demo landing page.	false

The following table lists the elements that an element of type `launch` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
description	push:string	A short description of this launcher.	0	1
url	push:string	The URL at which a webpage associated with this publisher can be found.	1	1
icon	push:string	A URL or path at which an icon representing this launcher can be reached.	0	1

property

A publisher property.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true
type	push:string	An optional property type. Usage of this is implementation specific.	false

credentials

Credentials for server connection.

The following table lists the elements that an element of type `credentials` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
username	push:string	User name.	0	1
password	push:string	Password.	0	1

server

The following table lists the attributes that an element of type `server` can have:

Name	Type	Description	Required
name	push:string	Server definition name.	true

The following table lists the elements that an element of type `server` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
host	push:string	The host name or IP address of the server.	1	1
port	push:port	The port number that the server is listening on for publisher client connections from other publishers.	1	1
ssl	push:boolean	If this value is true, the connection to the server is a secure connection over SSL. In this case the specified port must represent an SSL client connector at the server. The keystore properties must also be supplied for secure connections. By default, this is false.	0	1
keystore-file-location	push:string	The path of the keystore file defining the SSL context. This is ignored if ssl=false, but mandatory if it is true.	0	1
keystore-password	push:string	The keystore password. This is ignored if ssl=false, but mandatory if it is true.	0	1
input-buffer-size	push:bytes	Specifies the size of the input buffer to use for the connection with the server. This is used to receive messages from the server. Set this to the same size as the output buffer used at the server.	1	1
output-buffer-size	push:bytes	The size of the output buffer to use for the connection with the server. This is used to send messages to the server. Set this to the same size as the input buffer used by the server.	1	1
fail-policy	push:string	This specifies what happens if the publisher fails to connect to the server. 'default' means that if unable to connect, no action is taken and it is the publisher's responsibility to handle this. 'close' means that if unable to connect to the server, the publisher closes. 'retry' means that if unable to connect, the connection is automatically retried at intervals as specified by the retry-interval property.	1	1
retry-interval	push:millis	If the fail-policy for a server is 'retry', this is the interval at which the connection to the server is retried. If this value is not specified, a default of 5s is used.	0	1
credentials	credentials	Credentials to use for the server connection. If this value is not specified, no credentials are passed on connection.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
queue-definition	push:string	Optional outbound queue definition for this server connection. The definition must exist in Server.xml. This defines the queue to use for outbound messages from the publisher to the server. If this value is not specified, the default queue definition in Server.xml is used.	0	1

web-server

A web server definition.

The following table lists the elements that an element of type `web-server` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
virtual-host	push:string	The name of the virtual host to deploy to. If this value is not supplied, default-files-default is used.	0	1
alias-file	push:string	The alias file to use for this publisher	1	1

Server.xml

This file specifies the schema for the server properties, as well as multiplexers, security, conflation, client queues, and thread pools.

server

All server properties

The following table lists the elements that an element of type `server` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
server-name	push:string	The server name is used to identify this server if running in a cluster. This is also used as a prefix for client IDs. If not specified, the local hostname is used.	0	1
max-message-size	push:bytes	The maximum message size in bytes. This defines the maximum message size (including headers) that can be handled. The greater of this value and the input-buffer-size set in Connectors.xml is used when setting the socket input buffer size.	1	1
default-load-message-capacity	push:bytes	The default capacity of a load message if not explicitly specified. If not supplied, a default of 4096 is used.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
default-delta-message-capacity	push:bytes	The default capacity of a delta message if not explicitly specified. If not specified, a default of 1024 is used.	0	1
message-length-size	push:int	Specifies the number of bytes utilized within message headers to accommodate the message length. This can have a value of 4, 2, or 1. This is a system-wide setting which must match for all components of a distributed Diffusion system. Client APIs automatically adapt to the size at the server they connect to. The size chosen depends upon the maximum message size that needs to be catered for. A value of 1 caters for messages up to 127 bytes in length (including headers), 2 caters for messages up to 32,767 bytes in length, and 4 can notionally cater for messages up to 2,147,483,647 bytes on length. If a value is not supplied, a default of 4 is used.	0	1
charset	push:string	The default character set to use for Diffusion message character conversions. See Java Encodings for the full list. If a value is not specified, a default of "UTF-8" is used.	0	1
multiplexers	multiplexers	Properties that define multiplexers and how they are used.	1	1
write-selectors	write-selectors	DEPRECATED: These configuration options are no-ops retained for backwards compatibility. All types of selector have now been unified. The selectors used by Diffusion can be configured using selector-thread-pool-definition.	0	1
security	security	Properties relating to security (optional).	0	1
conflation	conflation	Conflation policies and topic to policy mappings.	0	1
client-queues	client-queues	Definitions of client queues.	1	1
connection-timeouts	connection-timeouts	Timeout values relating to connections. If a value is not specified, defaults are used.	0	1
date-formats	date-formats	Date and time formats. If a value is not specified, default formats are used.	0	1
thread-pools	thread-pools	Definitions of thread pools	1	1
selector-thread-pools	selector-thread-pools	Definitions of thread pools	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
whois	whois	Definition of the WhoIs lookup service. If a value is not specified, no WhoIs service runs.	0	1
auto-deployment	auto-deployment	Automatic deployment properties (optional).	0	1
geo-ip	geo-ip	Properties relating to the Geo IP lookup facility. If a value is not specified, defaults are used.	0	1
usr-lib	usr-lib	User libraries (optional).	0	1
hooks	hooks	User hooks used in the server (optional)	0	1

multiplexers

Multiplexer definitions.

The following table lists the elements that an element of type `multiplexers` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
client	push:string	Name of the client multiplexer definition. If this is not specified, the first multiplexer defined is used.	0	1
multiplexer-definition	multiplexer-definition	Multiplexer definition.	1	unbounded

multiplexer-definition

Multiplexer definition.

The following table lists the attributes that an element of type `multiplexer-definition` can have:

Name	Type	Description	Required
name	push:string	The multiplexer name.	true

The following table lists the elements that an element of type `multiplexer-definition` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
size	push:positiveInt	This is the number of multiplexer instances that start in readiness for clients to be assigned to. If there are going to be a large number of users, increase this number. If a value is not specified, a default value equal to the number of available processors is used.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
thread-priority	push:threadPriority	This is the thread priority that the multiplexer threads run at. If a value is not specified, a default of 8 is used.	0	1
load-balancer	push:string	DEPRECATED: This is the load balancer to use for for assigning connecting clients to multiplexer instances. There are currently two implemented load balancers, 'RoundRobin' and 'LeastClients'. If a value is not specified, a default of 'RoundRobin' is used. In future, only the 'RoundRobin' load balancing policy will be provided. If you have found the 'LeastClients' algorithm useful, contact Push Technology Support and reference case 11098.	0	1
latency-warning	push:millis	Multiplexers are critical to the operation of Diffusion. If there are too many clients assigned to too few multiplexer instances, there is a possibility of message latency. This is an optional flag which can be set to issue a warning if the multiplexer instance is taking too long in its operational cycle (see ServerNotificationListener in the publisher API). If this value is 0, this feature is not enabled. If this value is not supplied, a default of 0 is used.	0	1
max-event-queue-size	push:positiveInt	This specifies the maximum size of the multiplexer event queue. This is the queue on which events from publishers are queued for multiplexers and the default value is normally more than adequate. If this queue fills, it can cause the publisher threads to block until they can enqueue events and in this case it might be necessary to increase the value. Typically, leave this value at the default of 128k.	0	1

write-selectors

DEPRECATED: These configuration options are no-ops retained for backwards compatibility. All types of selector have now been unified. The selectors used by Diffusion can be configured using selector-thread-pool-definition.

The following table lists the elements that an element of type `write-selectors` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
thread-priority	push:threadPriority	DEPRECATED: This configuration setting is a no-op.	0	1
size	push:positiveNonZeroInt	DEPRECATED: This configuration setting is a no-op.	0	1
timeout	push:millis	DEPRECATED: This configuration setting is a no-op.	0	1
load-balancer	push:string	DEPRECATED: This configuration setting is a no-op.	0	1
queue-size	push:positiveNonZeroInt	DEPRECATED: This configuration setting is a no-op.	0	1

hooks

User hooks used in the server.

The following table lists the elements that an element of type `hooks` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
startup-hook	push:string	This is the class name of a class that implements the interface com.pushtechnology.diffusion.api.publisher.ServerStartupHook. If specified, the hook is instantiated and the serverStarting method called when the server is starting, before the loading of publishers.	0	1
shutdown-hook	push:string	This is the class name of a class that implements the interface com.pushtechnology.diffusion.api.publisher.ServerShutdownHook. If specified, the hook is instantiated and the serverStopping method called when the server is stopping.	0	1

security

Server security properties.

The following table lists the elements that an element of type `security` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
authorisation-handler-class	push:string	This is the full name of a class, on the classpath, that implements the AuthorisationHandler interface in the Java publisher API. If specified, the handler is instantiated when the server starts and is called to authorize client connections, subscriptions, and fetch requests.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
authentication-handlers	authentication-handlers		0	1

authentication-handlers

Authentication handlers, in order of decreasing precedence. The authentication handlers are called to authenticate new connections and changes to the principal associated with a session. Authentication handlers are configured in precedence order. Authentication succeeds if a handler returns "allow" and all higher precedence handlers (earlier in the order) return "abstain". Authentication fails if a handler returns "deny" and all higher precedence handlers return "abstain". If all authentication handlers return "abstain", the request is denied. After the outcome is known, the server might choose not to call the remaining handlers.

The following table lists the elements that an element of type `authentication-handlers` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
authentication-handler	server-authentication-handler	An authentication handler hosted by the server.	0	unbounded
control-authentication-handler	control-authentication-handler	An authentication handler registered by a client.	0	unbounded
system-authentication-handler	system-authentication-handler	An authentication handler that uses the configured system authentication store to validate principals and to define an action for anonymous logins. The XSD does not prevent you from configuring the system authentication multiple times. However, the Diffusion server restricts this and will not start if you define the system authentication handler more than once.	0	unbounded

server-authentication-handler

An authentication handler hosted by the server. The handler is instantiated when the server starts.

The following table lists the attributes that an element of type `server-authentication-handler` can have:

Name	Type	Description	Required
class	push:string	The class attribute specifies the fully qualified name of a handler implementation class that implements the com.pushtechnology.diffusion.client.security.authentication.Authentication interface. The class must be available on the classpath.	true

system-authentication-handler

The system authentication handler uses the configured system authentication store to validate principals and to define an action for anonymous logins. If the system handler is specified then it will check if a principal is specified in the store and if so will validate its credentials against the store. The store may also specify additional assigned roles to be granted to a principal. If a principal is not specified in the store then the handler will abstain. The store may indicate whether to allow, deny or abstain for anonymous logins.

control-authentication-handler

Client sessions register control authentication handlers using an identifying name. A <control-authentication-handler> must be configured with a matching handler-name. Configure at most one <control-authentication-handler> for a handler-name.

The following table lists the attributes that an element of type `control-authentication-handler` can have:

Name	Type	Description	Required
handler-name	push:string	The handler name attribute must match the identifying name used by the client session to register a control authentication handler.	true

conflation

Conflation policies and topic to policy mappings.

The following table lists the elements that an element of type `conflation` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
default-conflation-policy	push:string	The default conflation policy. This specifies a conflation policy that is used for any topics that do not have explicit conflation policy mappings defined. If this is not specified, conflation does not occur for topics that do not have a policy mapping defined. If this value is specified, it must be the name of a defined policy.	0	1
conflation-policy	conflation-policy	Conflation policy.	0	unbounded
topic-conflation	topic-conflation	A mapping between a topic (or topic selector pattern) and a policy.	0	unbounded

conflation-policy

Conflation policy.

The following table lists the attributes that an element of type `conflation-policy` can have:

Name	Type	Description	Required
name	push:string	The conflation policy name.	true

The following table lists the elements that an element of type `conflation-policy` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
mode	push:string	The conflation mode. This can have the value 'replace' or 'append'. If a value is not specified, a default of 'replace' is assumed. The value 'replace' means that when a matching message is found, the message is replaced by the new (or merged) message in its current queue position. The value 'append' means that when a matching message is found, the message is removed from its current queue position and the new (or merged) message is appended to the end of the queue. This option preserves message ordering but there is the danger that messages are constantly sent to the end of the queue.	0	1
matcher	push:string	The full class name of a message matcher of type com.pushtechnology.diffusion.api.conflation.MessageMatcher. If a class is not supplied, a default matcher that matches by topic name is used.	0	1
merger	push:string	The full class name of a message merger of type com.pushtechnology.diffusion.api.conflation.MessageMerger. If a class is not supplied, no merging with the new message occurs. The new message either replaces the existing one or the existing one is removed and the new one appended to the end of the queue depending upon the mode.	0	1

topic-conflation

A mapping between a topic (or topic selector pattern) and conflation policy.

The following table lists the elements that an element of type `topic-conflation` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topic	push:string	The name of a topic or a topic selector pattern that indicates the topic or topics that the specified conflation policy is applied to.	1	1
policy	push:string	The name of a configured conflation policy that is applied to the specified topic or topics.	1	1

client-queues

Client queue definitions.

The following table lists the elements that an element of type `client-queues` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
default-queue-definition	push:string	The name of the queue definition to use by default. Connectors that do not explicitly specify a queue definition use the one specified here.	1	1
queue-definition	queue-definition	Queue definition.	1	unbounded

queue-definition

This defines the properties of a client queue.

The following table lists the attributes that an element of type `queue-definition` can have:

Name	Type	Description	Required
name	push:string	The queue definition name.	true

The following table lists the elements that an element of type `queue-definition` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
max-depth	push:positiveInt	The maximum depth of the queue. If the number of messages queued for a client exceeds this number, the server disconnects the client.	1	1
conflates	push:boolean	Specifies whether conflation is applied to all clients using this queue definition. If this value is not specified, conflation is not applied by default.	0	1
upper-threshold	push:percent	This specifies a percentage of the maximum queue size and if this value is reached then any listeners (see ClientListener in the publisher API) are notified. Notification occurs only once and does not occur again until the queue has returned to the lower threshold. If this value is not specified, no upper limit notification occurs.	0	1
lower-threshold	push:percent	This specifies a percentage of the maximum queue size and indicates the level at which listeners (see ClientListener in the publisher API) are notified after an upper limit notification has occurred and the queue size has dropped back to	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
		the specified lower limit. If this value is not specified, no lower limit notification occurs.		
auto-fragment	push:boolean	Deprecated since 5.1.6. Topic message fragmentation has been removed. This setting has no effect.	0	1

connection-timeouts

Connection-related timeouts.

The following table lists the elements that an element of type `connection-timeouts` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
write-timeout	push:millis	This is the time in milliseconds that is allotted to performing a single write to a client. If the write has not completed within this limit, the client is disconnected. If this value is not specified, a default of 2s is used.	0	1
connection-timeout	push:millis	This is the time in milliseconds allowed for a connection to take place and complete its handshake processing. If this value is not specified, a default of 2s is used.	0	1

date-formats

Date and time formats.

The following table lists the elements that an element of type `date-formats` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
date	push:string	The format used when displaying dates. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd" is used.	0	1
time	push:string	The format used when displaying times. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "HH:mm:ss" is used.	0	1
date-time	push:string	The format used when displaying date and time. Specify the format according to the Java SimpleDateFormat	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
		specification. If a format is not specified, a default of "yyyy-MM-dd HH:mm:ss" is used.		
timestamp	push:string	The format used when displaying a timestamp - for example, in a log - to millisecond precision. Specify the format according to the Java SimpleDateFormat specification. If a format is not specified, a default of "yyyy-MM-dd HH:mm:ss.SSS" is used.	0	1

thread-pools

Thread pools.

The following table lists the elements that an element of type `thread-pools` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
inbound	push:string	Name of the inbound thread pool definition.	1	1
outbound	push:string	This property is deprecated. Do not use.	0	1
background-thread-size	push:int	Number of threads to use for the background thread pool. If a value is not specified, a default of 10 is used.	0	1
thread-pool-definition	thread-pool-definition	Thread pool definition.	1	unbounded

thread-pool-definition

Thread pool definition.

The following table lists the attributes that an element of type `thread-pool-definition` can have:

Name	Type	Description	Required
name	push:string	Name of the thread pool definition.	true

The following table lists the elements that an element of type `thread-pool-definition` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
core-size	push:positiveInt	The core number of threads to have running in the thread pool. Whenever a thread is required a new thread is created until this number is reached even if there are idle threads already in the pool.	1	1

Name	Type	Description	Minimu occurre	Maximum occurences
max-size	push:positiveInt	The maximum number of threads that can be created in the thread pool before tasks are queued. Such threads are released immediately after execution.	1	1
queue-size	push:positiveInt	The thread pool queue size. When the max-size is reached, tasks are queued. If the value is 0, the queue is unbounded. If the value is not 0, it must be at least 10.	1	1
keep-alive	push:millis	The time to keep inactive threads alive for. This does not apply to core threads. If this value is not specified, a default of 0 is used.	0	1
priority	push:threadPriority	This is the priority at which the threads run. If this value is not specified, a default of 5 is used.	0	1
thread-pool-listener	thread-pool-listener	Thread pool listener details (optional)	0	1
rejection-handler-class	push:string	The name of a class implementing the ThreadPoolRejectionHandler interface which is called if a task cannot be executed by the Thread Pool. If this value is not specified, a default rejection policy is used so that rejected tasks are executed in the calling thread. The default rejection policy is implemented by the class com.pushtechnology.diffusion.api.threads.ThreadService.CallerRunsReje A thread is rejected if all the threads are in use and the queue is full.	0	1

thread-pool-listener

Thread pool listener details.

The following table lists the elements that an element of type `thread-pool-listener` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
queue-notification-handler-class	push:string	The name of a class implementing the ThreadPoolNotificationHandler interface which is instantiated to handle notifications on the thread pool.	1	1
queue-upper-threshold	push:percent	The size of the thread pool queue at which the notification handler is called on the queueUpperThresholdReached method. The method is called once only until the queue size drops below the specified lower threshold.	1	1

Name	Type	Description	Minimu occurre	Maximum occurences
queue-lower-threshold	push:percent	The size of the thread pool queue at which the notification handler is called on the queueLowerThresholdReached method if the upper threshold has previously been breached.	1	1

selector-thread-pools

Thread pools.

The following table lists the elements that an element of type `selector-thread-pools` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
default	push:string	Name of the default selector thread pool definition.	0	1
selector-thread-pool-definition	selector-thread-pool-definition	Selector thread pool definition.	1	unbounded

selector-thread-pool-definition

Selector thread pool definition.

The following table lists the attributes that an element of type `selector-thread-pool-definition` can have:

Name	Type	Description	Required
name	push:string	Name of the selector thread pool definition.	true

The following table lists the elements that an element of type `selector-thread-pool-definition` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
size	push:positiveInt	The number of selector threads to have running in the thread pool. The number of selector threads created is the maximum of the value defined here and the number of acceptors defined in the Connectors.xml file. This number is fixed and does not change at runtime.	0	1

whois

WhoIs service details.

The following table lists the elements that an element of type `whois` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
provider	push:string	Name of the WhoIs provider class that must be on the classpath and must implement the API class WhoIsProvider. If a provider is not specified, WhoIsDefaultProvider is used.	0	1
threads	push:int	The number of background threads that process WhoIs resolver requests. If a value is not specified, a default of 2 is used. If the value is set to 0, the service is not started.	0	1
host	push:string	The hostname of a WhoIs provider that adheres to the RFC3912 WhoIs protocol. If a hostname is not specified, a default of "whois.ripe.net" is used.	0	1
port	push:port	The port number that the WhoIs provider listens on. If a value is not specified, the normal value of 43 is used.	0	1
whois-cache	whois-cache	Details of the WhoIs service cache that is used to cache WhoIs lookup results. If a value is not specified, the default values are used.	0	1

whois-cache

Details of the WhoIs service cache that is used to cache WhoIs lookup results.

The following table lists the elements that an element of type `whois-cache` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
maximum	push:int	The maximum size of the WhoIs cache. When the cache size exceeds this number it is tidied. A value of 0 means the cache grows indefinitely unless entries are removed because they have exceeded their retention time. If a value is not specified, a default of 1000 is used.	0	1
retention	push:millis	The time for which WhoIs cache entries are retained before being deleted. A value of 0 means entries are retained indefinitely or until the cache reaches its maximum size. If a value is not specified, a default of 0 is used.	0	1
tidy-interval	push:millis	The interval at which the Whois cache tidier task checks if any cache entries have passed their retention time or if the cache has exceeded its maximum	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
		size. This is ignored if both maximum and retention are 0. If a value is not specified, a default of 1 minute is used.		

auto-deployment

Auto deployment details.

The following table lists the elements that an element of type `auto-deployment` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
directory	push:string	The name of the automatic deployment directory.	1	1
scan-frequency	push:millis	The frequency at which the deployment directory is scanned for new deployments. If a value is not specified, a default of 5 seconds is used.	0	1

geo-ip

GeoIP details.

The following table lists the attributes that an element of type `geo-ip` can have:

Name	Type	Description	Required
enabled	push:boolean	Set to true to enable GeoIP lookup. This needs to be set to true if you are going to use connection or subscription validation policies. If a value is not specified, a default of true is used.	false

The following table lists the elements that an element of type `geo-ip` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
file-name	push:string	The name of the Maxmind GeoCityIP city file. If a value is not specified, a default of "../data/GeoLiteCity.dat" is used.	0	1

usr-lib

A list of user libraries from which user code is loaded.

The following table lists the elements that an element of type `usr-lib` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
directory	push:string	Directory to load classes from. When the server starts, this folder is traversed, including subdirectories and all jars or zip files added to the class loader.	1	unbounded

Related Links

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

User-written authentication handlers on page 145
You can implement authentication handlers that authenticate client connections to the Diffusion server or perform an action that requires authentication.

Statistics.xml

This file specifies the schema for the properties defining statistics collection. The statistics are broken into sections: client, topic, server and publisher.

statistics

Properties defining statistics collection.

The following table lists the attributes that an element of type `statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	A global switch to toggle collection of all statistics.	true

The following table lists the elements that an element of type `statistics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
client-statistics	client-statistics	Optional client statistics: configures Diffusion to periodically output client statistics to a log file defined in Logs.xml It gives a count of all of the different client types. Each counter is reset according to the configured frequency.	0	1
topic-statistics	topic-statistics	Optional topic statistics.	0	1
server-statistics	server-statistics	Optional server statistics.	0	1
publisher-statistics	publisher-statistics	Optional publisher statistics.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
reporters	reporters	Optional set of StatisticsReporters to be loaded with Diffusion, which are registered with the internal StatisticsService and used to generate output.	0	1

client-statistics

The following table lists the attributes that an element of type `client-statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies if aggregate client statistics are enabled.	true

The following table lists the elements that an element of type `client-statistics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
log-name	push:string	Definition of the log in Logs.xml.	0	1
output-frequency	push:millis	Specifies the output frequency of the log. There is one entry per specified interval. If this is not specified, a default of 1h is used.	0	1
reset-frequency	push:millis	Specifies when the counters are reset. Zero specifies that the counters are never reset. If this is not specified, a default of 1h is used.	0	1
monitor-instances	push:boolean	Specifies if individual client statistics are enabled.	0	1

topic-statistics

The following table lists the attributes that an element of type `topic-statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies if aggregate topic statistics are enabled.	true

The following table lists the elements that an element of type `topic-statistics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
monitor-instances	push:boolean	Specifies if individual topic statistics are enabled.	0	1

publisher-statistics

The following table lists the attributes that an element of type `publisher-statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies if aggregate publisher statistics are enabled.	true

The following table lists the elements that an element of type `publisher-statistics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
monitor-instances	push:boolean	Specifies if individual publisher statistics are enabled.	0	1

server-statistics

The following table lists the attributes that an element of type `server-statistics` can have:

Name	Type	Description	Required
enabled	push:boolean	Specifies whether to enable server statistics. This enables high-level aggregate statistics for the system.	true

reporters

The set of StatisticsReporters that the Diffusion server is aware of at startup. Used to output the statistics gathered for clients, publishers, or topics.

The following table lists the elements that an element of type `reporters` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
reporter	reporter	A reporter definition.	0	unbounded

reporter

A StatisticsReporter definition.

The following table lists the attributes that an element of type `reporter` can have:

Name	Type	Description	Required
name	push:string	The reporter name.	true
enabled	push:boolean	Whether the reporter is enabled. If this is set to true, the reporter is automatically loaded when Diffusion starts. Otherwise, you must manually load the reporter config at run-time using the statistics API.	true

The following table lists the elements that an element of type `reporter` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
type	push:string	The type of reporter to be used. Currently options are: TOPIC - exposes metrics in the Diffusion topic tree; JMX - exposes metrics on the local JMX server.	1	1
property	property	A property available to the reporter. This can be used to configure reporter-specific variables or parameters.	0	unbounded

property

A StatisticsReporter property. Currently accepted values:
 'interval' - used by the topic reporter. Specifies an integer value used to set the period of update publishing.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	The property value	true
type	push:string	An optional property type. Usage of this is implementation specific.	false

SubscriptionValidationPolicy.xml

This file specifies the schema for the subscription validation policy.

subscription-validation-policies

Subscription validation policies

The following table lists the elements that an element of type `subscription-validation-policies` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topics	topics	A map of topics to policies.	0	1
policy	policy	A subscription validation policy.	0	unbounded

topics

A map of topics to policies.

The following table lists the elements that an element of type `topics` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
topic	topic	A topic to policy mapping.	0	unbounded

topic

The following table lists the attributes that an element of type `topic` can have:

Name	Type	Description	Required
policy	push:string	The name of the policy to apply to this topic.	true

policy

A subscription validation policy.

The following table lists the attributes that an element of type `policy` can have:

Name	Type	Description	Required
name	push:string	Each policy must be supplied with a unique name for easy reference.	true
type	push:string	The policy type is either "blacklist" or "whitelist". A blacklist indicates that if any of the policy rules in this policy match the incoming connection, that connection is to be rejected. A whitelist requires that at least one policy rule matches for the connection to be accepted.	true
automatic	push:boolean	Policies which are set to automatic are applied by Diffusion and the publishers do not need to perform any checks themselves. If this is set to false, the policy is not applied unless it is done by the publisher. If this value is not specified, a default of true is used.	false
validate-children	xsd:boolean	Controls whether to perform validation on child topics if the parent topic fails validation. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `policy` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
addresses	addresses	Subscription validation policy addresses. These are addresses that are blacklisted/ whitelisted.	0	1
locale	locale	Connection validation policy locale. This is locale details that are blacklisted/ whitelisted.	0	unbounded

addresses

The following table lists the elements that an element of type `addresses` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
address	push:string	An IP address (or regular expression) of a subscribing client.	0	unbounded

Name	Type	Description	Minimu occurre	Maximum occurences
hostname	push:string	The hostname (or regular expression) of a subscribing client.	0	unbounded
resolved-name	push:string	The resolved hostname (or regular expression) of a subscribing client, as returned by the WhoIs service.	0	unbounded

locale

The following table lists the elements that an element of type `locale` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
country	push:string	The ISO country code of the subscribing client, as returned by the WhoIs service.	0	1
language	push:string	The ISO language code of the subscribing client, as returned by the WhoIs service.	0	1

WebServer.xml

This file specifies the schema for the web server properties.

web-servers

Definitions of one or more web servers.

The following table lists the elements that an element of type `web-servers` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
web-server	web-server	Web server definition.	0	unbounded

web-server

Web server definition.

The following table lists the attributes that an element of type `web-server` can have:

Name	Type	Description	Required
name	push:string	Name of the web server definition.	true

The following table lists the elements that an element of type `web-server` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
client-service	client-service	Optional client service.	0	1
http-service	http-service	HTTP service.	0	unbounded
file-service	file-service	Optional file service.	0	1

virtual-host

Virtual host definiton.

The following table lists the attributes that an element of type `virtual-host` can have:

Name	Type	Description	Required
name	push:string	Virtual host name.	true
debug	push:boolean	Debug flag. Set to true for debugging. Default is false.	false

The following table lists the elements that an element of type `virtual-host` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
host	push:string	Specifies the host which the virtual host is to serve, for example, download.pushtechnology.com or * for all.	1	1
document-root	push:string	The physical directory for this virtual host. If a relative path is configured, it is resolved relative to the Diffusion home directory.	1	1
home-page	push:string	The default home page. This file is used with directory browsing.	1	1
error-page	push:string	This is used to control the 404 response. The server looks for one of these files in the directory of the request. If the file does not exist, it looks for this file in the virtual directory. If the file is not supplied or the file does not exist, a standard 404 response HTML document is sent.	0	1
static	push:boolean	If this is set to true, after loading the resource once, the file system is not checked again. This improves performance for simple static usage. By default this is false.	0	1
minify	push:boolean	Set to true to minify the html. This happens before the file is compressed. By default this is false.	0	1
cache	cache	The virtual host cache configuration.	1	1
compression-threshold	push:bytes	All HTTP responses over this size are compressed. If not specified, a default value of 512 is used.	0	1
alias-file	push:string	Optionally specifies an alias file. This allows for URL aliasing if required. If a relative path is configured, it is resolved relative to the Diffusion configuration directory.	0	1
realms	realms	Virtual host realms.	0	1

realms

Virtual host realms.

The following table lists the elements that an element of type `realms` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
realm	realm	A virtual host realm.	0	unbounded

realm

A virtual host realm.

The following table lists the attributes that an element of type `realm` can have:

Name	Type	Description	Required
name	push:string	Virtual host realm name.	true
path	push:string	Virtual host realm path.	true

The following table lists the elements that an element of type `realm` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
users	users	Virtual host realm users.	0	1

users

Virtual host realm users.

The following table lists the elements that an element of type `users` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
user	user	Virtual host realm user.	1	unbounded

user

Virtual host realm user.

The following table lists the attributes that an element of type `user` can have:

Name	Type	Description	Required
name	push:string	Virtual host realm user name.	true
password	push:string	Virtual host realm user password.	true

cache

Virtual host cache.

The following table lists the attributes that an element of type `cache` can have:

Name	Type	Description	Required
debug	push:boolean	Set true to debug the cache. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `cache` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
file-size-limit	push:bytes	If the file to be served is over this size, do not cache the entire contents, but map the file instead. If a size is not specified, a default value of 1m is used.	0	1
cache-size-limit	push:bytes	Total size of the cache for this web server definition. If a size is not specified, a default value of 10m is used.	0	1
file-life-time	push:millis	If the file has not been accessed within the time specified, remove the entry from the cache. If a time is not specified, a default value of 1d is used.	0	1

http-service

HTTP service.

The following table lists the attributes that an element of type `http-service` can have:

Name	Type	Description	Required
name	push:string	HTTP service name.	true
debug	push:boolean	Set true to debug the HTTP service. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `http-service` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
class	push:string	The user HTTP service class name. This class must implement the HTTPServiceHandler interface in the web server API.	1	1
url-pattern	push:string	The pattern that the URL must match for this service to be invoked.	1	1
log	push:string	An optional log file can be specified and, if so, HTTP access can be logged. The log definition must exist in Logs.xml.	0	1
max-inbound-request-size	push:bytes	The maximum number of bytes that the HTTP request can have. If this is not specified, a default of the maximum message size is used.	0	1
property	property	HTTP service property.	0	unbounded

property

A property.

The following table lists the attributes that an element of type `property` can have:

Name	Type	Description	Required
name	push:string	Property name.	true
type	push:string	Optional property type.	false

file-service

File service.

The following table lists the attributes that an element of type `file-service` can have:

Name	Type	Description	Required
name	push:string	File service name.	true

The following table lists the elements that an element of type `file-service` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
virtual-host	virtual-host	Virtual host.	1	unbounded
write-timeout	push:millis	Write timeout for serving files. This does not affect HTTP clients. If a value is not specified, a default value of 3s is used.	0	1

client-service

Client service.

The following table lists the attributes that an element of type `client-service` can have:

Name	Type	Description	Required
name	push:string	Client service name.	true
debug	push:boolean	Set true to debug the client service. If a value is not specified, a default of false is used.	false

The following table lists the elements that an element of type `client-service` can contain:

Name	Type	Description	Minimu occurre	Maximum occurences
message-sequence-timeout	push:millis	This is used with HTTP clients to indicate how long to wait for a missing message in a sequence of messages before assuming it has been lost and closing the client connection. If a value is not specified, a default of 2s is used.	0	1
websocket-origin	push:string	This is used to control access from client web socket to Diffusion. This is a regular expression pattern that matches the origin of the request. A value of ".*" matches anything, so all requests are allowed. If this is not specified, the service is unable to handle web socket requests.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
cors-origin	push:string	This is used to control access from client web (XHR) to Diffusion. This element will enable Cross Origin Resource Sharing (CORS). This is a regular expression pattern that matches the origin of the request. A value of ".*" matches anything, so all requests are allowed. If a value is not specified, the service cannot handle CORS requests.	0	1
websocket-secure-response	push:boolean	Deprecated - the value is ignored. This configuration applied to behavior defined by draft versions of the WebSocket specification that is no longer supported.	0	1
close-callback-requests	push:boolean	For Diffusion client requests this specifies whether to obey the keep-alive header or close all requests. If this is set to true, all requests are closed. If a value is not specified, a default of false is used.	0	1
compression-threshold	push:bytes	Enable compression for HTTP client responses over this size. If a value is not specified, a default of 512 bytes is used.	0	1
max-inbound-request-size	push:bytes	The maximum number of bytes that the HTTP request can have. If a value is not specified, a default of the maximum message size is used.	0	1
comet-bytes-before-new-poll	push:bytes	This parameter enables you to specify the number of bytes after which a Comet connection is forced to re-establish itself. This can help to reduce the potential for memory leaks in the browser due to the long-lived nature of a Comet connection, at the expense of degraded performance due to more frequent HTTP handshakes. If this is not specified, a default value of 30 kilobytes is used.	0	1
comet-initial-message-padding	push:bytes	Some browsers do not pass on data received through a Comet connection until a minimum number of bytes have been received. This means that in the case where an initial topic load message is small, the client might never receive it. To work around this restriction, you can set a value here which ensures that the first message received is padded with extra bytes that are automatically discarded by the client library. If a value is not specified, a default of 1k is used.	0	1

Name	Type	Description	Minimu occurre	Maximum occurences
disable-cookies	push:boolean	Set true to disable session cookie from being in the "Set-Cookie" header. If a value is not specified, cookies are enabled.	0	1

Additional XML files

The `etc` directory contains additional XML files. The format of these XML files is not defined by Push Technology.

The following XML files are included in the `etc` directory. Their use is described in other sections of this manual.

`crossdomain.xml`
`FlashMasterPolicy.xml`
`FlashPolicy.xml`

Use these files to grant a web client permission to handle data across multiple domains.

`JMSAdapter51.xml`

Use this file to configure the legacy JMS adapter version 5.1.

Related Links

Cross domain

Server

The Diffusion server is the core component of the Diffusion Product.

The server is written in Java and can be deployed on any platform that supports Java.

Related Links

Installing on page 560
You can install the Diffusion server from a JAR file or through Red Hat Package Manager.

Server basics

What is the Diffusion server?

The Diffusion server is a highly efficient and low latency messaging server which provides the functions listed here.

- A scalable and mature solution to push (stream) and receive data and events, in real-time, both to and from a web browser and other net-connected devices.
- High performance transport of data between two or more machines within your own network or extranet.
- Web server capability.

A single server can be deployed to push messages to web browsers (or other net devices) or many interconnected servers can provide a fully scalable enterprise messaging solution.

Starting the server

Here we learn the basic of how to configure and start a Diffusion Server.

Before you begin

A Diffusion server instance can be installed as described in the Installing on page 560. After installing, the installed files are present in the specified install directory. As issued, a Diffusion server is configured to run the demo publishers and these can be tried out and explored to familiarize with the product.

However, for a real implementation it is first necessary to design and write publisher and client applications.

Once a publisher is made available, Diffusion can be configured to run it. This is done by editing the XML property files. For more information, see XML configuration on page 573.

To configure the server to run a user-written publisher involves the following:

Procedure

1. Define connectors appropriate for the types of connections (client, policy file, and so on) that are supported using `etc/Connectors.xml`
2. Define publishers using `etc/Publishers.xml`.

   Other configuration might be required according to the nature of the application. The above steps are the bare minimum requirement to get a publisher working.

   The server produces logs which record actions performed by the server and can also be used for diagnostic purposes. The level of logging produced and where logging is directed is configurable. For more information, see Logging on page 721.
3. The `diffusion.sh` or `diffusion.bat` command (issued in the `bin` directory) starts Diffusion. An optional properties directory can be specified as a parameter to be used instead of the default `../etc` directory.

   > **Important:** Do not run your Diffusion server as root on Linux or UNIX. To run the Diffusion server on a port number of 1024 or lower, use another means. For some examples of ways of doing this, see http://www.debian-administration.org/articles/386.

   Once a server is started, it must be accessible from client applications or test tools.
4. A running server can be managed using JMX or using the management API.
5. Tuning the server is critical to getting the best performance out of a Diffusion installation.

   Some of the aspects that must be considered are message design and sizing, Concurrency, client queues, and connection buffers. For more information, see Tuning on page 704.

Running from within a Java application

To run Diffusion from within a Java application instantiate, configure and start a `DiffusionServer` object.

Creating a server

`DiffusionServer` is available in the `com.pushtechnology.diffusion.api.server`. You can instantiate it with one of the following constructors:

Default configuration

```
DiffusionServer server = new DiffusionServer();
```

This instantiates the server with default configuration options. The default configuration is read from the XML configuration files in the `etc` directory of your Diffusion installation. Required aspects of the server must be configured before it is started.

Bootstrap properties

```
DiffusionServer server = new
 DiffusionServer(bootstrapProperties);
```

This specifies a set of properties inside a `Properties` object. The following properties are supported:

Property	Description	Default
diffusion.home	The base installation directory	Calculated from the location of the `diffusion.jar` file.
diffusion.config.dir	The configuration directory, where the XML configuration files are located.	*`diffusion.home`*`/etc`
diffusion.license.file	The license file	*`diffusion.config.dir`*`/licence.lic`
diffusion.keystore.file	The keystore file required to decrypt the license	*`diffusion.config.dir`*`/licence.keystore`

These properties can also be set as system properties.

Whichever approach to instantiation that you use, a full set of XML configuration files can be present in the configuration directory and tuned as required or just a partial set of the files can be present and all missing configuration supplied programmatically.

Configuring the server

Once the server object has been instantiated it can be configured. The root configuration object can be obtained from the server object as follows:

```
ServerConfig config = server.getConfig();
```

Alternatively the root can be obtained using `ConfigManager.getServerConfig()`.

The configuration is populated from the XML configuration files in the installation or in the configuration directory specified by *diffusion.config.dir*. You can further customize the configuration to fit your requirements. For an example, see the following sample code:

```
DiffusionServer server = new DiffusionServer();

ServerConfig config = server.getConfig();

// Publisher
PublisherConfig publisher = config.addPublisher("My
 Publisher","com.company.MyPublisherClass");

// Connector ConnectorConfig connector =
 config.addConnector("Client Connector");
// Configure connector as required....

// Thread Pools
ThreadsConfig threads = config.getThreads();
ThreadPoolConfig inbound = threads.addPool("Inbound");
inbound.setCoreSize(3);
inbound.setMaximumSize(10);
inbound.setQueueSize(2000);
inbound.setPriority(8);
threads.setInboundPool(inbound.getName());
threads.setBackgroundPoolSize(2);

// Queues QueuesConfig queues = config.getQueues();
QueueConfig queue = queues.addQueue("DefaultQueue");
queue.setMaximumDepth(10000);
queues.setDefaultQueue("DefaultQueue");

// Multiplexer MultiplexerConfig multiplexer =
 config.addMultiplexer("Multiplexer");
multiplexer.setSize(4);
```

Starting the server

After the server configuration has been completed, the server can be started using `server.start()`.

The declared publishers are then loaded and connectors start to listen on the configured ports.

Stopping the server

The server can be stopped using `server.stop()` at which point the server is no longer available.

Run requirements

A simple way to use Diffusion as a library within your application is to install Diffusion and include the path to `diffusion.jar` in your CLASSPATH.

If you want to repackage the Diffusion code more extensively, be aware of the following concerns:

- `diffusion.jar` depends on other library files in the installation's `lib` directory and are referenced in the jar's manifest `Class-Path` entry. You must also make the code in these libraries available.

- You still require a Diffusion installation. The installation provides the configuration, licence, and log directories. The installation directory is calculated from the location of `diffusion.jar`. If `diffusion.jar` is not loaded from a URL classloader, or has been moved from the product installation, use the bootstrap properties constructor and set the *diffusion.home* system property to the installation directory.

Limitations

Currently only one Diffusion server can be instantiated in a Java VM and it can be started only once.

Concurrency

Diffusion is a multi-threaded server and utilizes concurrent processing to achieve maximum performance. Java NIO technology is utilized so that a separate thread is not required for each concurrent connection and very large numbers of concurrent connections can be handled.

Because Diffusion is a multi-threaded environment it is necessary to have an understanding of concurrency issues when writing publishers and when configuring Diffusion for best performance. For more information, see Tuning on page 704.

This section discusses issues of threading and concurrent processing.

Publisher threads

The processing that occurs within the user-written code of a publisher can be executed in different threads as discussed below. Any publisher method can be called at the same time as another. Because of this all publisher processing must be thread safe and it is the user's responsibility to synchronize processing as required. It is recommended that synchronization is maintained at the smallest scope possible to avoid performance bottlenecks.

Inbound threads

Any input that is received on an NIO connection is processed by a thread from the inbound thread pool . This includes most publisher notifications from clients, event publishers or other publishers with the exception of control notifications (such as `initialLoad`, `publisherStarted`) which occurs in the controlling thread.

> **Note:** The act of publishing or sending messages to clients is asynchronous that is to say that the message is queued for the client or clients. Publisher processing is not blocked whilst messages are delivered to clients. For best performance it is recommended that any code executed in the inbound threads is non-blocking (for example, avoid database access, locking, and disk IO as much as possible).

Client notification threads

If a publisher uses client notifications, the publisher has its own dedicated thread to process those notifications.

By default here is one notification thread per publisher, no matter how many listeners are defined. Each event is processed by the thread in the order in which they occur and two client notification event methods are not called concurrently. If the order of such events is not critical, you can specify that a user thread pool is used for client notifications this increasing throughput.

User threads

Publishers or other users of the Diffusion Java API can make use of the Java threads API to schedule tasks for processing of their own in a separate thread of processing.

You can execute any object of a class that implements the `RunnableTask` interface using one of the `ThreadService.schedule` methods. You can to request a one-off execution of a task, periodic execution at a given interval or execution according to a schedule. Periodic processing can be important to publishers that pull data updates from elsewhere.

Such tasks issued using the thread service are executed using threads from the background thread pool.

Alternatively, users can define their own thread pools to use using the thread service and execute tasks using these thread pools.

NIO Threads

Each connector that is configured in `etc/Connectors.xml` comprises a connector thread that listens for incoming socket connections, accepts them and registers them with an acceptor thread that handles any incoming data notifications. Message decoding, routing to publishers and appropriate publisher callbacks are all run in the inbound thread pool. Connector and acceptor threads are occupied for the minimum amount of time and are completely non-blocking.

Though performance can be improved in extreme case by adjusting the numbers of these NIO threads, no significant processing occurs within them.

Client multiplexers

A client multiplexer is a separate thread which is responsible for processing messages on the publisher event queue, queuing for clients (conflating if necessary), taking messages from client queues and sending them to the client or clients. A number of these multiplexers can be configured to improve concurrent processing when there are a large number of clients.

The number of multiplexers can be configured. By default, the number of multiplexers is the same as the number of available cores on the host system of the Diffusion server.

Multiplexers typically batch these output messages into output buffers according to the output buffer size configured for the client connectors.

Thread pools

Diffusion maintains a number of configurable thread pools which are used for a number of purposes

For more information, see Thread pools on page 708. Thread pools can also be accessed programmatically using the `ThreadService` class of Diffusion server API. Refer to the API documentation for more information about this.

The various types of thread pools are as follows:

Inbound thread pool

This is used to obtain a thread to process any inbound message received on an NIO connection. The maximum number of threads configured for this pool must cater for the maximum required concurrency for incoming requests.

Diffusion does not maintain a separate thread for each client connection but rather passes each inbound request from a connection to the inbound thread pool for processing.

For example, when a client subscribes, the input processing happens on an inbound thread from the pool, the subscribe method and topic loader methods are run in one of these threads.

Connector inbound thread pools

Individual connectors can configure their own separate inbound thread pool to override the use of the default. This cannot be required if you want different behaviors for each connector or if there are a lot of connectors. Due to locking on the inbound thread pool, it is more performant for each connector to have its own inbound thread pool.

Background thread pool

The background thread pool is used for executing scheduled tasks. These tasks can be issued by Diffusion itself or using a publisher using the Java threads API.

Diffusion uses scheduled tasks for various reasons such as:

1. Timing out ACK messages. A scheduled task executes if a message sent to a client is not acknowledged within its required timeout period.
2. Retrying connections. If a Diffusion server cannot connect to another server and there is a retry policy, a scheduled task will be used to retry the connection.

If any publisher uses a lot of scheduled tasks, the number of threads in this pool might have to be increased waiting tasks might queue.

Unlike other types of pool when the specified number of threads are in use, tasks are queued in an unbounded queue.

User thread pools

Within the Java threads API user can define thread pools that can be used for multi-threaded processing.

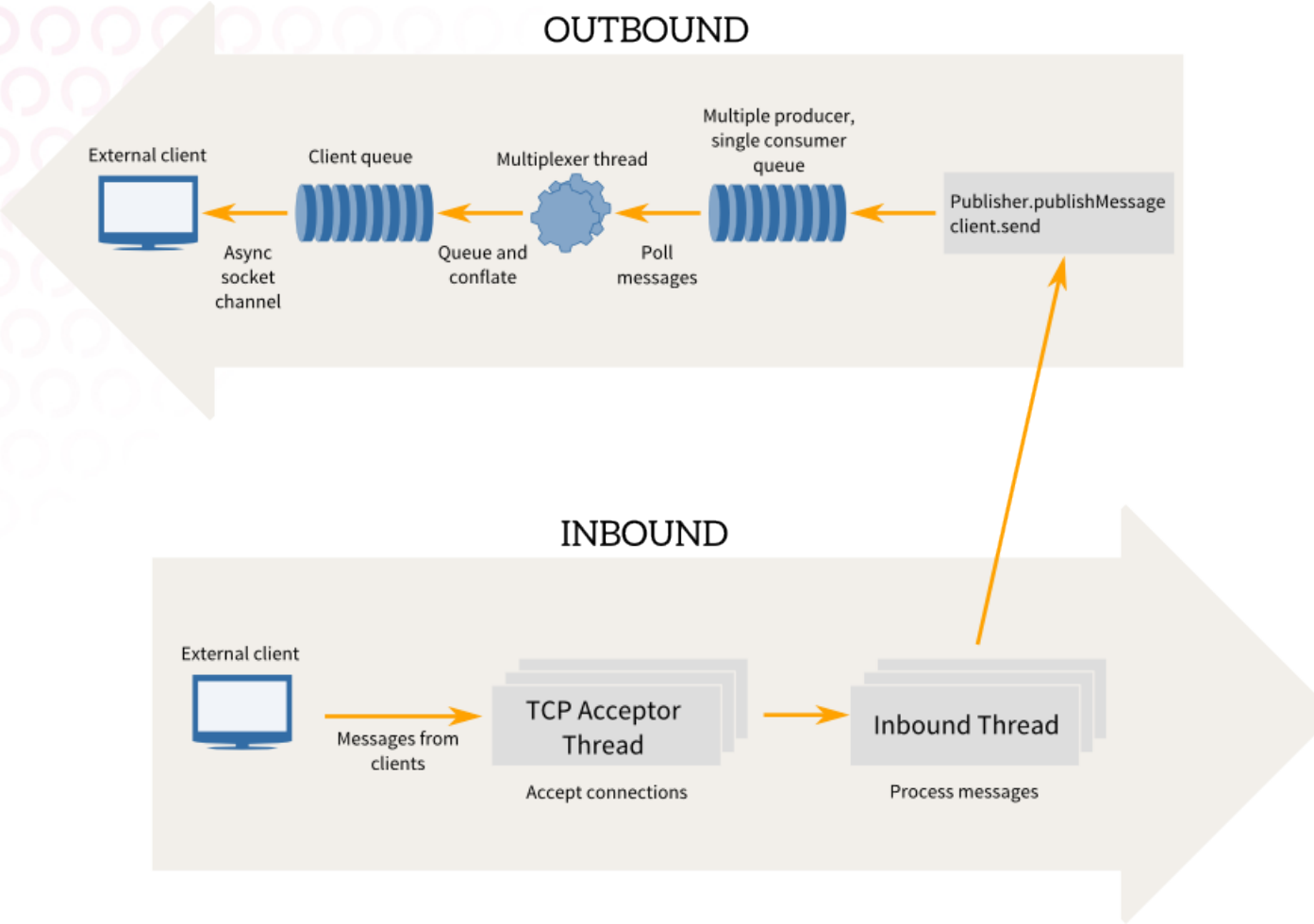


Figure 57: Thread diagram

Connectors

An introduction to the concept of connectors within a Diffusion server.

A connector provides a connection point for external applications to connect to a Diffusion server over a TCP connection.

Each connector has a 'Socket Server' thread which reacts to an incoming connection. The socket information is defined by the connector.

Suitable connectors must be defined for inbound connections expected by a server. Connectors are defined in the `etc/Connectors.xml` configuration file.

The following properties are common to all connectors:

Table 79: Connectors properties

Name	A name by which the connector can be identified.
Port	A port number on which to accept requests (or policy requests).
Host	The host to accept requests (only relevant on a multi-homed machine).

DEPRECATED: Number of acceptor threads	The number of acceptor threads. This can be tuned for performance reasons.
Input buffer size	The size of the socket input buffer to use for each connection.
Output buffer size	The size of the socket output buffer to use for each connection.

Socket buffer sizes are very important in achieving the best performance. For more information, see Tuning on page 704.

Restricting connection types

By default a connector can accept any type of connection handled by Diffusion, but you can configure a connector so that it accepts only one type of connection as follows:

Table 80: Connection restrictions

client	Client connections.
policy	Policy file requests.
all	Any type of connection.

Client connections

Connectors can accept connections from any type of client. Any number of connectors can be defined to provide different connection points with different properties.

Each client connection has an input buffer to receive messages from the client. The configured input buffer size must be large enough to accommodate the largest message expected from the client. If the maximum message size and the input buffer size are configured as different values, the larger of the two is used as the input buffer size.

The output buffer size is used to assign an output buffer per client multiplexer into which messages are dequeued prior to transmission. The output buffer size must be at least as large as the largest message that is sent to the client (typically the largest initial topic load message). However, a much bigger size can be configured so that messages can be batched for sending to the client. Ideally the buffer size is a multiple of the average message size to allow for suitable batching. This can have an important effect on performance. For more information, see Tuning on page 704.

Note: The buffer size must allow extra for header information when using HTTP.

Enabling client reconnection

Specify how long a disconnected client's session is kept alive on the server and how many messages to queue for a client while it is disconnected by using the `<reconnect>` element in the `etc/Connectors.xml` configuration file.

```
<connector>
 ...
 <reconnect>
  <keep-alive>60s</keep-alive>
  <max-depth>1000</max-depth>
 </reconnect>
 ...
</connector>
```

A client can reconnect to the server through this connector within 60 seconds of becoming disconnected. While the client is disconnected, up to 1000 messages are queued for it. These messages are delivered to the client when it reconnects.

Restricting client connections by API

Connectors can accept connections from clients that use either the Unified API or Classic API. Specify which API the connector accepts connection through by using the `<api-type>` element in the `etc/Connectors.xml` configuration file.

Policy connections

Connectors are used to serve policy file requests to plugin clients. Flash and Silverlight require policy files to be served from different ports (Flash on 843 and Silverlight on 943) so if plugin clients are in use it will be necessary to define a separate connector for each type of plugin client.

The connector configuration specifies the path of an XML policy file to be sent to the client.

Related Links

Connectors.xml on page 607
This file specifies the schema for the connectors properties.

Connectors.xml on page 607
This file specifies the schema for the connectors properties.

Load balancers

Diffusion is commonly used with load balancers such as F5 Networks Inc.'s BIG-IP suite, to distribute incoming requests fairly over a number of Diffusion servers.

Each client is registered against a single Diffusion instance which contains client that is not replicated to other instances. For this reason, it is necessary for all traffic to and from that client is routed to the correct Diffusion server.

For HTTP-based protocols there might be many connections established, dropped and re-established during the lifetime of a single client session. Every one of these must be routed to the correct Diffusion server.

At first glance, streaming protocols such as DPT that open a single socket and remain connected until they are no longer required appear immune to requiring any special considerations. However, in the event that connection keep-alive is enabled to handle reconnections in case of temporary connection loss, it is important that the reconnection attempt is routed to the original server.

HTTP protocols

To route HTTP traffic, the load balancer must be able to inspect the HTTP headers and extract session information. Diffusion sets an HTTP cookie named `session` with a connection-specific ID specifically for this purpose. Load balancers typically have facilities for using this to maintain a table of client to server mappings.

Sample HTTP conversation, cookie highlighted:

```
POST /diffusion/ HTTP/1.1
Host: localhost:8080
```

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:22.0)
 Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
m: 0
ty: B
t: null
tt: 90
v: 4
username: null
password: null
Referer: http://localhost:8080/tools/DhtmlClient.html
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
Cookie: GUID=95UAaOVCK6FPFB2J9dBl
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

HTTP/1.1 200 OK
Set-Cookie: session=clyde-r6t80mwyjav6
Cache-Control:no-store, no-cache
Content-Type:text/plain; charset=UTF-8
Content-Length:26

4.100.4.clyde-r6t80mwyjav6

POST /diffusion/ HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:22.0)
 Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
m: 1
c: clyde-r6t80mwyjav6
Referer: http://localhost:8080/tools/DhtmlClient.html
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
Cookie: session=clyde-r6t80mwyjav6; GUID=95UAaOVCK6FPFB2J9dBl
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
```

If you want to configure your load balancer to inject its own cookie, you can disable this Diffusion cookie. Disabling the Diffusion cookie avoids the following drawbacks:

- The overhead of transmitting the cookie
- The overhead of parsing the cookie
- The overhead of storing a per-session mapping from the Diffusion cookie value to the Diffusion server
- The risk of the browser exposing the session ID/token over HTTP

To disable the Diffusion cookie, set the `<disable-cookie>` element in the `WebServer.xml` configuration file of your Diffusion to `true`.

Streaming protocols

Without the ability to parse headers (and indeed, the absence of a session ID at all), the most common method for routing a streaming protocol such as DPT or WebSockets is to create a client/server mapping based on the IP addresses of the endpoints. This technique is generally referred to as Sticky-IP, and has the advantage of also working with HTTP transports, if required.

For F5's Sticky IP, ensure that the Source Address Translation option is set to Auto Map.

Figure 58: Sticky-IP in F5 BIG-IP

The drawback of this approach is that multiple users masquerading behind a proxy or access point can have the same IP address, and all requests from clients with that IP address are routed to the same Diffusion instance. Load balancing still occurs, but some hosts might be unfairly loaded.

Routing strategies

Load balancers often present several different strategies for choosing which server is associated with a new request. Common examples are:

Table 81: Routing strategies

Name	Description
Round-robin	Each available Diffusion instance is chosen in turn, with none favored.
Fewest clients	The server with the fewest number of client connections in progress is chosen.
Least loaded	The server with the lowest CPU load is chosen.

Monitors

Determining the availability of a Diffusion node can be achieved in more than one way. Commonly, an HTTP probe is used against the built-in web server. This has the advantage of being simple; most system administrators are familiar with HTTP requests. In the simplest case, a GET request can be made against the root context of the web server, for example:

```
GET / HTTP/1.0\r\n
```

However, this only tests the availability of the Diffusion server as a whole, and not the applications within it. A more advanced approach is to implement an `http-service`, as documented in the web-servers section. This queries the state of the publishers or the topic tree, and return availability on receipt of a GET request.

A better, but more complex approach is to implement a custom monitor using a scripting language that is supported by the load balancer. In the case of BIG-IP, this is a custom Diffusion client written in Perl that connects and subscribes to a status topic.

> **Note:** Push Technology does not provide or support custom monitors written by third parties.

Connection pooling

Many load balancers include a connection pooling feature, such as the BIG-IP OneConnect profile, where connections between the load balancer and the Diffusion server are kept alive and reused by other clients. In fact, multiple clients can be multiplexed through a single server side connection.

In Diffusion, a client is associated with a single TCP/HTTP connection for the lifetime of that connection. If a Diffusion server closes a client, the connection is also closed. Diffusion makes no distinction between a single client connection and a multiplexed connection, so when a client sharing a multiplexed connection closes, the connection between the load balancer and Diffusion is closed, and subsequently all of the client-side connections multiplexed through that server-side connection are closed.

For this reason, it is required that load balancers are not configured to pool connections when working with Diffusion.

Further reading

F5/BIG-IP:F5: Enabling Session Persistence

Web server

Diffusion incorporates its own web server.

Any Diffusion connector can be configured to act as a web server and service Diffusion clients, file requests, or any request for which a user-defined HTTP service has been configured.

Interaction with publishers

Describes how publishers can interact with the web server

Server-side processing

Server-side processing can be utilized with any file that has a text mime type and JavaScript. Currently there are three server-side tags, Include, Publisher and Topic Data. These tags are stored in HTML comments so as to not interfere with normal HTML

Include Tag

Include stubs load the file specified in the file attribute and are loaded as is into the parent HTML document. They do not necessarily have to be valid HTML. They can be positioned anywhere within the HTML file.

These includes are synonymous with `#Include` statements of ANSI C.

Below is an example of the syntax:

```
<!--@DiffusionTag type="Include" file="stub.html" -->
```

Include files can be nested so an include file can contain an include tag

Publisher tag

Publisher tags enable a publisher to interact with the web page during the serving process. Again these tags can appear anywhere within the HTML document. In the case below the publisher method `processHTMLTag` of the `Trade` publisher is called with the tag argument of `table` The publisher can return a String of HTML that is inserted into the document at the

position of the tag and the tag is removed. The `processHTMLTag` method is also called with the HTTP Request, although the request cannot be written to. Below is an example of the syntax

```
<!--@DiffusionTag type="publisher" publisher="Trade"
 tagid="table" -->
```

TopicData

Topic data tags allow for `SingleValueTopicData` items to be rendered in the HTML page. Again these tags can appear anywhere within the HTML document. The following example shows the syntax:

```
<!--@DiffusionTag type="TopicData" name="Assets/FX/EURUSD/O" -->
```

HTTP listener

Publishers can listen to all file HTTP requests by registering as a `HTTPRequestListener`. This exposes the interface

```
void handleHTTPRequest(HTTPVirtualHost virtualHost,HTTPRequest
 request)
```

This enables for more detailed statistics to be captured from the HTTP request

Security

Aspects of web server security

Digest authentication

Digest authentication can be utilized to negotiate credentials with a user's web browser. It is applied to specific directories on your web site. The protection of one directory automatically applies protection to all lower directories as well. The configuration file `etc/WebServer.xml` is used to add new realms and to store the user's name and the passwords.

HTTP deployment

You can deploy DAR files to a Diffusion server through a web service. This web service does not run by default, but can be enabled for your test environment by editing the `WebServer.xml` configuration file.

> **Warning:** Access to the deploy web service is not restricted. Do not enable this web service in your production environment unless you restrict access to the *`diffusion-url`*`/deploy` URL by other means, for example through your firewall setup.

System management

This section discusses how to manage a Diffusion server and system as a whole.

Going to production

When going to production with Diffusion review the information in this section.

Before going to production

Security

Develop the handlers that you require secure your Diffusion solution. You can use handlers in the Diffusion server to manage authentication and authorization.

For more information, see User access control.

Ensure that you change the default security settings in the `etc/SystemAuthentication.store` file. This file contains well known usernames and passwords, provided for developer convenience, that compromise the security of your solution if left unedited.

Testing

Thoroughly test your Diffusion solution before deploying it in production.

Diffusion provides a number of tools that you can use to test your solution. For more information, see Testing on page 542.

Performance

Understand how your Diffusion solution performs.

You can use the benchmarking suite (available on GitHub) to gauge the performance of your Diffusion solution. For more information, see Benchmarking suite on page 558.

You can view metrics and statistics for your Diffusion solution through the Publisher API or through MBeans. For more information, see Statistics.xml on page 637.

Tuning

Tune your Diffusion solution to ensure optimum performance for your expected load and usage patterns.

For more information, see Tuning on page 704.

Putting your Diffusion solution into production

Diffusion and your network

- Load balancers on page 657
- Network security on page 702

In production

Monitoring

Monitor your Diffusion solution using the monitoring console.

For more information, see Diffusion monitoring console on page 688.

Diagnostics

If an issue occurs in production, you can use log out put and other diagnostics to understand and fix the issue.

For more information, see Diagnostics on page 721.

General management

You can start the Diffusion server by using one of the provided scripts.

Procedure

The Diffusion server is started using the `diffusion.bat` (or `diffusion.sh` on UNIX platforms) script in the issued `bin` directory.

Configuration files are loaded by default from the `etc` directory. An optional properties directory can be specified as a parameter to the `diffusion` command. If a properties directory is specified, any properties files in the specified directory are loaded in preference to those in the `etc` directory. The server logs where it loads each property file from.

Classic deployment

Installing publishers into a stopped Diffusion instance.

The publishers that are started when Diffusion starts must be defined in the configuration file `etc/Publishers.xml`. Publishers that do not start with Diffusion can also be defined in the `etc/Publishers.xml` and these can be started later using JMX.

The publishers must be present on the classpath of the Diffusion server. The recommended way to do this is to compile the publisher source code with the `diffusion.jar` they run with on the classpath. Package the publisher class files into a JAR file. This JAR file must be deployed to the `ext/` directory of the Diffusion installation. The Diffusion server will search the `ext/` directory and load all the JAR files it finds.

Related Links

Defining publishers
Writing a publisher on page 153
How to approach writing a publisher

Hot deployment

Installing publishers into a running Diffusion instance.

In addition to starting publishers by defining them in `etc/Publishers.xml`, you can install them into an already running Diffusion instance by a process known as hot deployment. Publishers can also be undeployed and redeployed, providing they implement the `isStoppable` method, and it returns true. You can also deploy dependent JAR files, configuration files and associated web pages for a publisher. All artifacts required for deployment are packaged within a DAR file.

What's in a DAR file?

A DAR file contains the JAR file that contains the publisher Java files, one or more XML configuration files, a manifest file, and any other files required by your publisher.

Figure 59: Example folder structure inside a DAR file

The root folder name is the name of the publisher.

The `MANIFEST.MF` file contains an attribute, `Diffusion-Version`, which specifies the minimum version number of Diffusion on which this publisher runs. This prevents deployment of publishers to Diffusion instances which might not support features of the publisher or have different API signatures.

```
Manifest-Version: 1.0
Diffusion-Version: 5.5.3
```

The `etc` directory contains files which are normally found in a Diffusion installation's `etc` directory, but contain only information relating to the publisher being deployed. Files that affect the operation of Diffusion and have no relationship to the publisher are not loaded. Valid configuration files are:

`etc/Aliases.xml`

Include this file if there are associated HTML files.

`etc/Publishers.xml`

You must include this file.

`etc/SubscriptionValidationPolicy.xml`

Include this file if it is referenced from the `etc/Publishers.xml` file.

The `Publishers.xml` file has the same structure and the one in a Diffusion installation's `etc` directory. For more information, see Publishers.xml on page 618.

For example:

```
<publishers>
        <publisher name="MyPublisher">

 <class>com.pushtechnology.diffusion.test.publisher.MyPublisher</
class>
                <start>true</start>
                <enabled>true</enabled>
        </publisher>
</publishers>
```

Put all Java code required by your publisher in the `ext` folder. All files in this directory are placed on the classpath for the publisher. You can include any required third-party JAR files or resources in this folder.

Building a DAR file

You can use the Maven plugin *mvndar* to build and deploy your publisher DAR file. Alternatively, you can use an Ant build file to perform the same function.

Building a DAR file using mvndar (recommended)

The mvndar plugin for Maven is developed and maintained on GitHub: https://github.com/pushtechnology/mvndar.

To use mvndar, reference it in the `plugins` section of your `pom.xml` file.

```
<plugins>
      ...
      <plugin>
        <groupId>com.pushtechnology.tools</groupId>
        <artifactId>dar-maven-plugin</artifactId>
        <version>1.2</version>
        <extensions>true</extensions>
      </plugin>
      ...
    </plugins>
```

The plugin runs during the project's package phase and creates the DAR file in the target directory.

Building a DAR file using Apache Ant

The following example `build.xml` for Ant takes a publisher developed as an Eclipse project, packages it in a temporary directory, and copies it to the directory where the Ant script is executed:

```
<project name="MyPublisher" default="maketmpable">
    <property name="publisher.name" value="MyPublisher" />
    <property name="jar.name" value="${publisher.name}.jar" />
    <property name="diffusion.dir" value="." />
    <property name="dar.name" value="${publisher.name}.dar" />

    <target name="makejar">
        <jar jarfile="${jar.name}" includes="**/*.class"
 basedir="." />
    </target>

    <target name="maketmpable" depends="makejar">
        <tempfile property="temp.file"
 destDir="${java.io.tmpdir}" prefix="publisher-name"/>
        <copy todir="${temp.file}/${publisher.name}">
            <fileset dir=".">
                <include name="etc/**" />
                <include name="ext/**" />
                <include name="html/**" />
                <include name="META-INF/**" />
            </fileset>
        </copy>
        <copy todir="${temp.file}/${publisher.name}"
 file="${jar.name}" />
```

```
        <jar jarfile="${dar.name}" includes="${publisher.name}/
**" basedir="${temp.file}" manifest="META-INF/MANIFEST.MF" />
        <delete dir="${temp.file}" />
    </target>
</project>
```

Related Links

Using Maven to build Java Diffusion applications on page 786
Apache Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

Building publishers and other server application code with Maven on page 788
The Diffusion API for publishers and other server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Deployment methods

There are two ways to deploy a DAR file: file copy or HTTP.

File copy

To use this method, copy your DAR file to the deployment directory on the file system. If you enable auto-deployment in the `Server.xml` configuration file, Diffusion periodically scans a directory for new or updated DAR files and deploys them. In the case of an updated DAR, the existing publisher is undeployed (if possible) before being redeployed.

HTTP

If the deploy web service is running, you can POST the DAR file over HTTP. For example, you can use command line tools such as curl to deploy the publisher:

```
curl --data-binary @MyPublisher.dar http://localhost:8080/deploy
```

> **Warning:** We recommend you use the HTTP method of deployment in your test environments only. If you enable the deploy web service in your production environment, you must take additional security measures to prevent unauthorized or malicious access to the web service URL. For example, by setting up restrictions in your firewall.

To enable deployment through HTTP, you must enable the web service in the `WebServer.xml` configuration file. For example, include the following XML:

```
<http-service name="deploy-service" debug="true”>
 <class>com.pushtechnology.diffusion.service.DeploymentService</
class>
 <url-pattern>^/deploy.*</url-pattern>
 <max-inbound-request-size>128m</max-inbound-request-size>
</http-service>
```

Ensure that the HTTP connector is configured to have an input buffer large enough to contain the entire DAR file. You can configure this in the `Connectors.xml` configuration file.

Undeployment

For publishers deployed using the file copy method, you can delete the DAR file from the deployment directory and on the next scan the server undeploys the publisher. A DAR file can be undeployed only if all of the publishers it contains are stoppable. If a DAR file fails to be undeployed, any future modifications to it are ignored.

It is important that any files that the deployment process has extracted from the DAR are not deleted until the publisher has been successfully undeployed. Publishers can also be undeployed through JMX by invoking the undeploy operation on associated MBean, for example

```
localhost/Server/com.pushtechnology.diffusion - Publisher -
 MyPublisher - undeploy()
```

Using JMX

You can use JMX to manage Diffusion. By default, the RMI registry port is 1099 and the JMX service port is 1100.

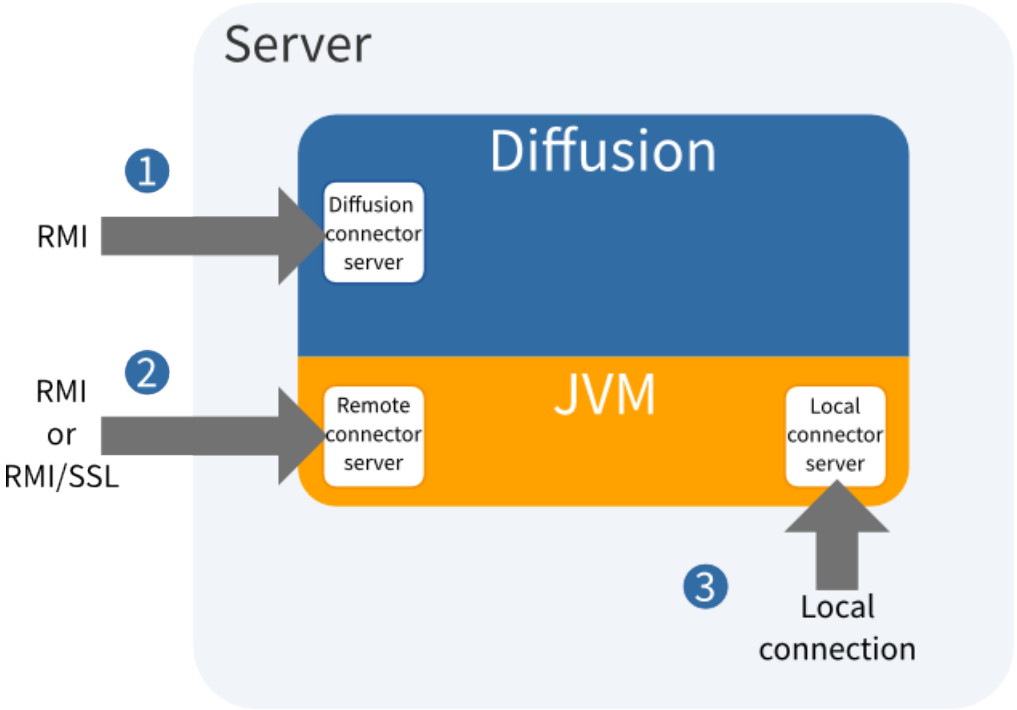


Figure 60: Connecting to Diffusion JMX

The following methods of connecting to Diffusion JMX are available:

1. **Recommended:** Through the RMI JMX connector server provided by the Diffusion server.

   This feature is integrated with Diffusion security, enabling you to use roles and permissions to control access to the MBeans. However, this connection does not use SSL.

   For more information, see Configuring the Diffusion JMX connector server on page 668.
2. Through the RMI JMX connector server provided by JVM that runs Diffusion.

   You can use SSL to make a secure connection. However, the JVM does not use Diffusion security. You must add additional configuration to your JVM to control access to the MBeans.

For more information, see Configuring a remote JMX server connector on page 670.

3. Through the local JMX connector server provided by JVM that runs Diffusion.

   You make this connection from the server that Diffusion runs on. However, the JVM does not use Diffusion security. You must add additional configuration to your JVM to control access to the MBeans.

   For more information, see Configuring a local JMX connector server on page 671.

Related Links

Management.xml on page 616
This file specifies the schema for the management properties that enable JMX access over an RMI JMXConnectorServer.

Configuring the Diffusion JMX connector server

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

About this task

Diffusion binds to the specified ports to listen for connections from JMX clients such as JConsole and Java VisualVM.

Procedure

1. Optional: If you are running Diffusion on a Linux server, check that the host name is not `127.0.1.1`.

   You can do this my running the following command:

   ```
   hostname -i
   ```

   If the output to this command is `127.0.1.1`, add an entry to `/etc/hosts` that defines the host name.

2. Edit the `etc/Management.xml` configuration file to enable and configure the management features:

   a) Set the value of the `enabled` attribute in the `management` element to `true`.

   ```
   <management enabled="true">
   ```

   b) Optional: Specify the ports to use for the JMX service.

   ```
       <!-- The RMI Registry port -->
       <registry-port>1099</registry-port>
       <!-- The JMX service port -->
       <connection-port>1100</connection-port>
   ```

   If the Diffusion server runs on JDK 7u4 or later, these two ports can be set to the same value. This can simplify firewall configuration.

   You can use the default values:

   - **1099** The RMI registry port
   - **1100** The JMX service port

c) Specify the roles that are assigned to authenticated principals.

```
<assigned-roles>
  <role>OPERATOR</role>
</assigned-roles>
```

These roles are assigned to all principals that connect through the Diffusion JMX connector server. The roles a principal has determine the actions the principal can take on the Diffusion server.

For read-only access to the JMX service a user must have the view_server permission. In the default security configuration the OPERATOR role gives the principal the view_server permission.

3. Configure the principals that are allowed to use the JMX service. You can do this in one of the following ways.
   - Update the system authentication store to assign a role with the required permissions to the principal and configure the Diffusion server to call the system authentication handler.

     For more information, see System authentication handler on page 147.
   - Implement a custom authentication handler that assigns a role with the required permissions to the principal and configure the Diffusion server to call your custom authentication handler.

     For more information, see User-written authentication handlers on page 145.
   - The JMX users can be set in the `etc/Management.xml` file, but this capability is deprecated. Instead use authentication handlers to decide which principals are allowed to connect and to assign them the appropriate roles.

     Users defined in this way that are not marked as read-only, are assigned the role JMX_ADMINISTRATOR which contains the permissions required for write access to the JMX service.
4. **Note:** If you are using a firewall that employs NAT, you might still be unable to connect to Diffusion even when the JMX ports are left open.

   Optional: To make a secure connection or a connection through a firewall, you can use SSH tunnelling:
   a) Establish an SSH connection to the fire-walled Diffusion server.
   b) Tunnel the RMI registry port and JMX service port through SSH.
   c) Use JMX to connect to the local ends of the tunneled ports.

Results

Use the ports you have configured to connect a JMX management console to the Diffusion server.

This connection cannot be made through SSL. However, you can use SSH tunnelling to secure your connection. For more information, see step 4 on page 669.

Related Links

Using Java VisualVM on page 671
You can manage Diffusion using the JMX system management console Java VisualVM.

Using JConsole on page 673

You can manage Diffusion using the JMX system management console JConsole.

Configuring a remote JMX server connector

Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

About this task

Important: We recommend that you use the Diffusion connector server to connect to the JMX service. For more information, see Configuring the Diffusion JMX connector server on page 668.

The JVM that runs Diffusion accepts remote connections from JMX clients such as JConsole and Java VisualVM.

Procedure

1. Configure security for your remote JMX connection.

   For more information, see https://docs.oracle.com/javase/jp/8/technotes/guides/management/agent.html.

   The security users and roles defined for the JVM do not integrate with the security provided by the Diffusion server
2. When starting Diffusion, set the properties required for your remote JMX connection.

   For more information, see https://docs.oracle.com/javase/jp/8/technotes/guides/management/agent.html.
3. **Note:** If you are using a firewall that employs NAT, you might still be unable to connect to Diffusion even when the JMX ports are left open.

   Optional: To make a secure connection or a connection through a firewall, you can use SSH tunnelling:

   a) Establish an SSH connection to the fire-walled Diffusion server.
   b) Tunnel the RMI registry port and JMX service port through SSH.
   c) Use JMX to connect to the local ends of the tunneled ports.

Results

Use the ports you have configured to connect a JMX management console to the Diffusion server. These connections can be made over SSL.

Related Links

Using Java VisualVM on page 671
You can manage Diffusion using the JMX system management console Java VisualVM.

Using JConsole on page 673

You can manage Diffusion using the JMX system management console JConsole.

Configuring a local JMX connector server

Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

About this task

The JVM that runs Diffusion accepts local connections from JMX clients such as JConsole and Java VisualVM.

Procedure

> **Note:** If you are using Java version 7 or later, you do not have to take any steps to configure the JVM to provide a JMX service.

Review the JVM documentation for any actions to take before connecting your JMX client.

For more information, see https://docs.oracle.com/javase/jp/8/technotes/guides/management/agent.html.

Results

You can connect a JMX management console running on the same server as Diffusion to the JVM.

Related Links

Using Java VisualVM on page 671
You can manage Diffusion using the JMX system management console Java VisualVM.

Using JConsole on page 673
You can manage Diffusion using the JMX system management console JConsole.

Using Java VisualVM

You can manage Diffusion using the JMX system management console Java VisualVM.

Java VisualVM is usually installed with JDK's but can be downloaded from https://visualvm.dev.java.net/.

Connecting to the Diffusion connector server

1. Start Java VisualVM.
2. Right-click on the **Remote** section of the **Applications** panel and select **Add Remote Host**.
3. In the **Host name** field, type the host name or IP address of the server where Diffusion is running. Click **OK**.
4. In the **Applications** panel, right-click on the name of the server where Diffusion is running. Select **Add JMX Connection**.
5. In the **Connection** field, enter the host name and RMI registry port for the Diffusion server.
6. Select **Use security credentials** and enter the username and password of a principal that you have configured to be able to use JMX. For more information, see Configuring the Diffusion JMX connector server on page 668. Click OK.

Information about the Diffusion process is displayed in the main panel.

Connecting to the JVM remote connector server

Note: We recommend you use the Diffusion connector server to access the JMX service.

1. Start Java VisualVM.
2. Right-click on the **Remote** section of the **Applications** panel and select **Add Remote Host**.
3. In the **Host name** field, type the host name or IP address of the server where Diffusion is running. Click **OK**.
4. In the **Applications** panel, right-click on the name of the server where Diffusion is running. Select **Add JMX Connection**.
5. In the **Connection** field, enter the host name and RMI registry port for the Diffusion server.
6. Select **Use security credentials** and enter the username and password of a user that you have configured in the JVM. For more information, see Configuring a remote JMX server connector on page 670. Click OK.

Information about the Diffusion process is displayed in the main panel.

Connecting to the JVM local connector server

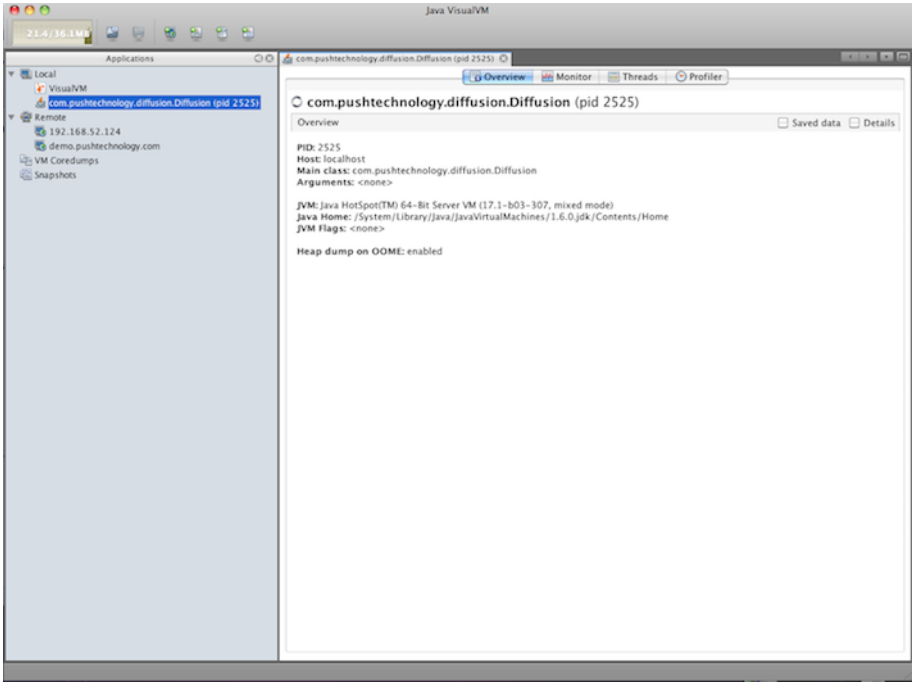


Figure 61: Java VisualVM: Overview tab

Note: We recommend you use the Diffusion connector server to access the JMX service.

1. Start Java VisualVM.
2. From the **Local** section of the **Applications** panel, select the Diffusion process, `com.pushtechnology.diffusion.Diffusion`.
3. Right-click `com.pushtechnology.diffusion.Diffusion` and select **Open**.

Information about the Diffusion process is displayed in the main panel.

Once connected to JMX, several aspects of the system are available to monitor and tune. For more information, see the Java VisualVM documentation: http://visualvm.java.net/docindex.html.

Related Links

Configuring the Diffusion JMX connector server on page 668
Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

Configuring a local JMX connector server on page 671
Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Configuring a remote JMX server connector on page 670
Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Using JConsole

You can manage Diffusion using the JMX system management console JConsole.

Connecting to the Diffusion connector server

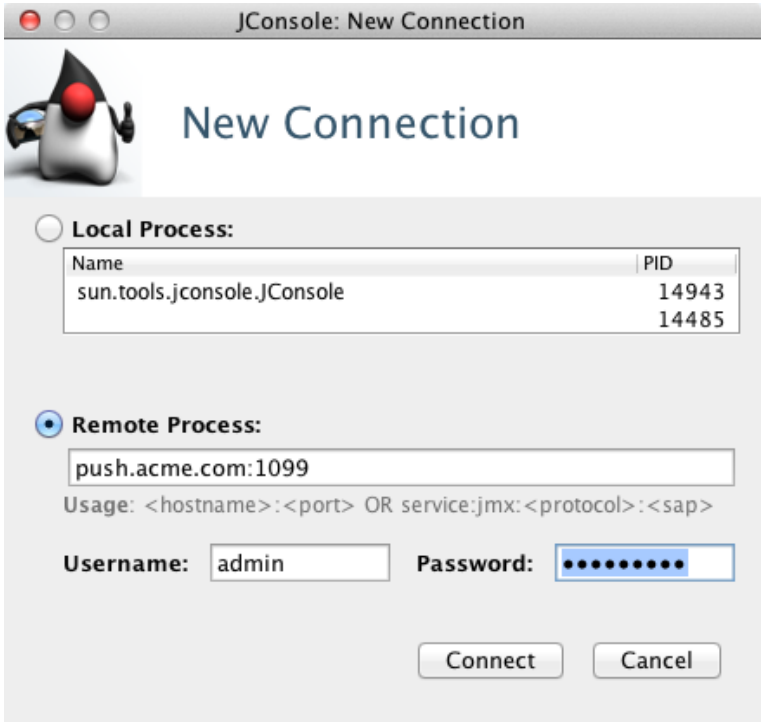


Figure 62: JConsole New Connection dialog: Remote Process

In the **Remote Process** section of JConsole's **New Connection** dialog, enter the following information:

- The host name and RMI registry port of the Diffusion connector server. The default RMI registry port is 1099.
- The username and password of a principal that you have configured to be able to use MBeans. For more information, see Configuring the Diffusion JMX connector server on page 668

Connecting to the JVM remote connector server

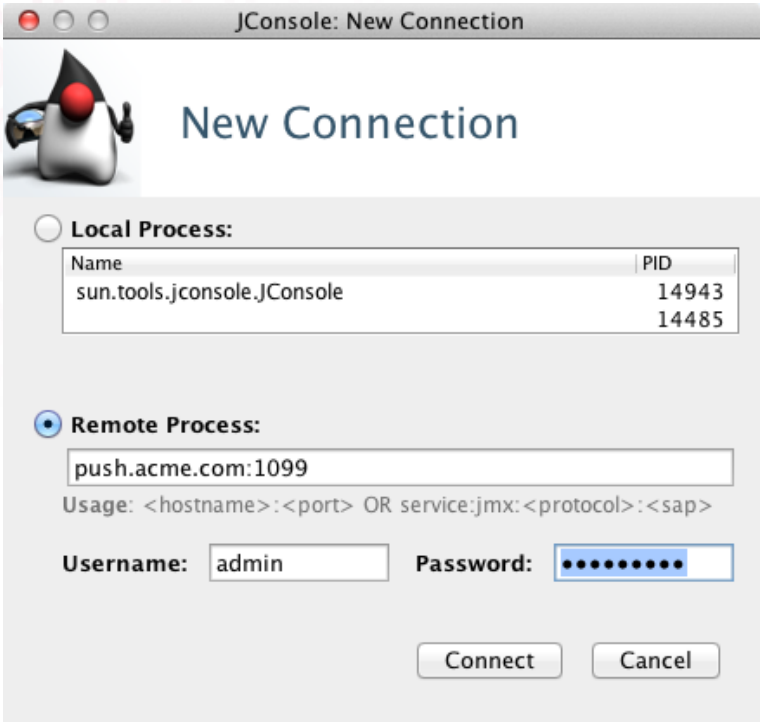


Figure 63: JConsole New Connection dialog: Remote Process

Note: We recommend you use the Diffusion connector server to access the JMX service.

In the **Remote Process** section of JConsole's **New Connection** dialog, enter the following information:

- The host name and RMI port of the Diffusion connector server. The default port is 1099.
- A username and password that you have configured in the JVM to be able to connect to the JMX service. For more information, see Configuring a remote JMX server connector on page 670

Connecting to the JMX service

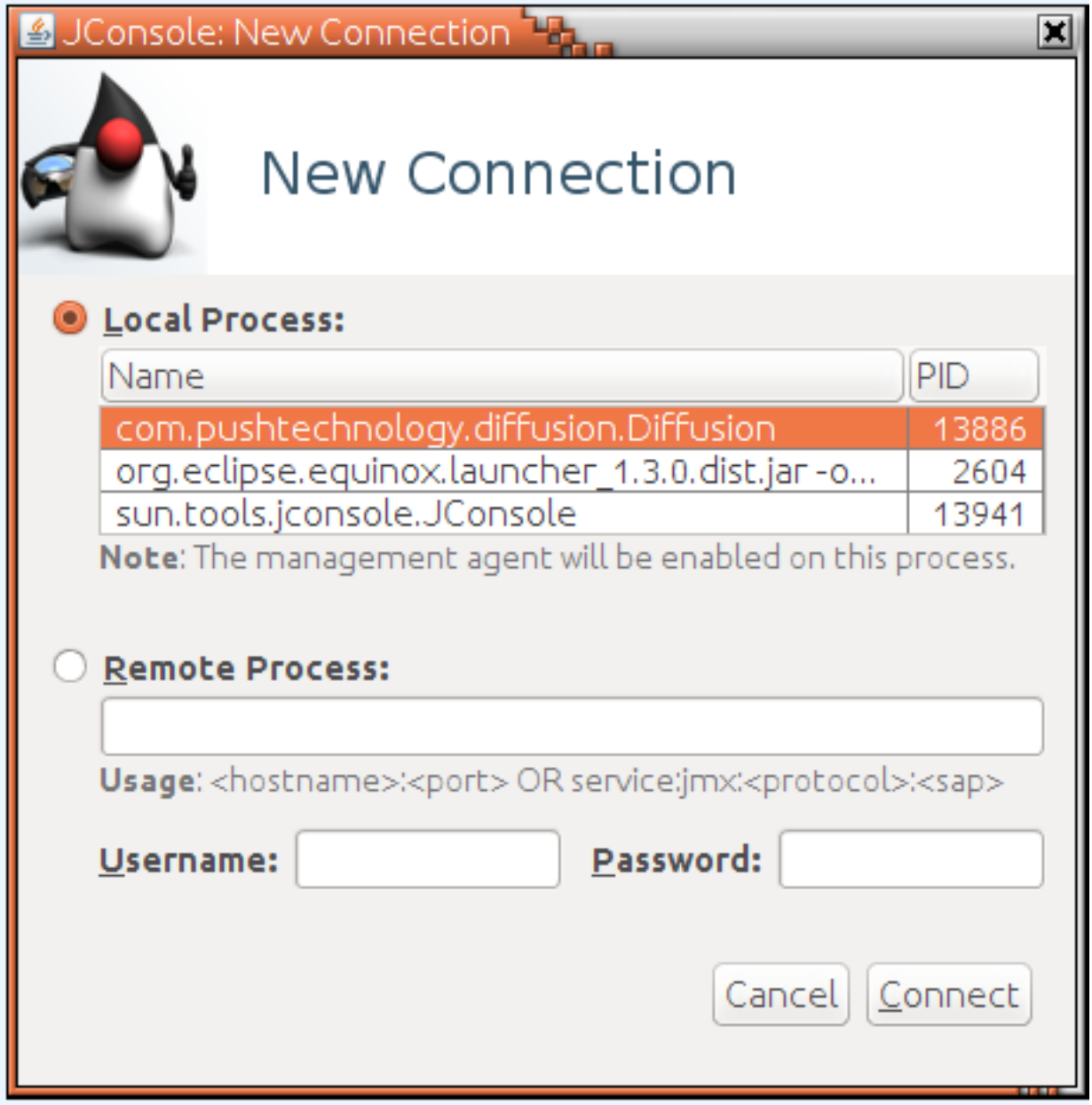


Figure 64: JConsole New Connection dialog: Local Process

Note: We recommend you use the Diffusion connector server to access the JMX service.

In the **Local Process** section of JConsole's **New Connection** dialog, select the Diffusion process, `com.pushtechnology.diffusion.Diffusion`.

Once connected to JMX, several aspects of the system are available to monitor and tune. For more information, see the JConsole documentation: https://docs.oracle.com/javase/jp/8/technotes/guides/management/jconsole.html.

Related Links

Configuring the Diffusion JMX connector server on page 668

Connect to JMX through the Diffusion connector server. This connector server is integrated with the Diffusion server and enables you to use role-based access control to define how connecting users can use the MBeans.

Configuring a local JMX connector server on page 671
Connect to JMX through a local connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Configuring a remote JMX server connector on page 670
Connect to JMX through a remote connector to the JVM that runs the Diffusion. This connector is not integrated with the Diffusion server security and you must configure additional security in the JVM.

Detecting deadlocks with JConsole

To check if your publisher is experiencing a deadlock, you can use JConsole to inspect the threads.

Procedure

1. Connect JConsole to the JMX service on the Diffusion server.

   For more information, see Using JConsole on page 673.
2. Go to the **Threads** tab.

   This tab provides information about thread use: number of threads, a list of active threads, and details for a selected thread.
3. On the **Threads** tab, click the **Detect Deadlock** button.

   If deadlocks are present, new tabs open next to the **Threads** tab. You can use the information in these tabs to diagnose any deadlocks.

What to do next

For more information, see http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html.

MBeans

Diffusion registers MBeans with the JMX service for many of its principal features.

Annotations on each of the MBeans employed are used to produce the following pages in this manual as well as feeding JMX clients with descriptive information. MBeans, attributes and operations have descriptions; operation arguments have names; operations also have JMX impact information.

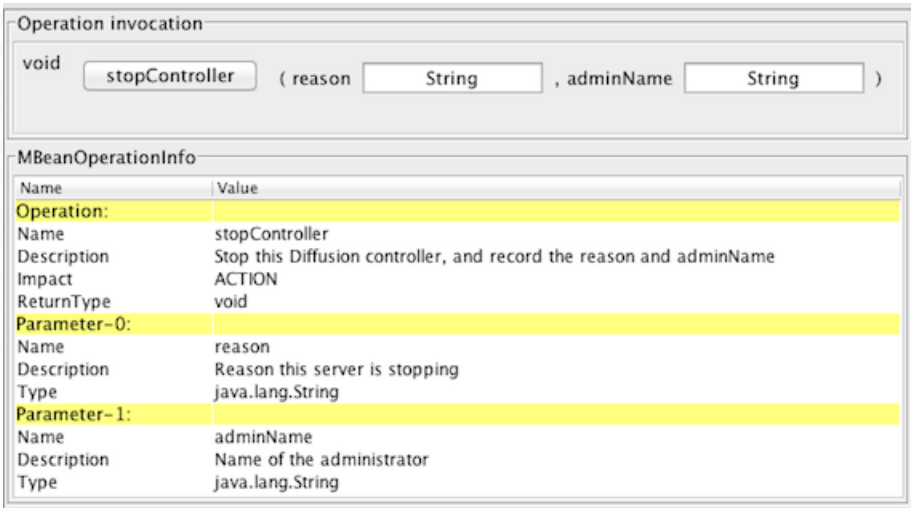


Figure 65: The server MBean stopController operation showing in JConsole

AggregateStatisticsMBean

Interface for a specific StatisticsCollector

Attributes

Name	Type	Read/Write	Description
instanceStatisticsEnabled	boolean	read-write	Whether statistics collection is enabled or not.
managementName	String	read	A name describing the set of statistics being aggregated.

AuthorisationManagerInstanceMBean

Management interface to the optional AuthorisationManager

Attributes

Name	Type	Read/Write	Description
connections	int	read	Number of connections authorized
connectionsDenied	int	read	Number of connections denied authorization
fetches	int	read	Number of fetches authorized
fetchesDenied	int	read	Number of fetches denied authorization
handlerClassName	String	read	Class name of any configured AuthorisationHandler
hasHandler	boolean	read	True if this server has an AuthorisationHandler configured
subscriptions	int	read	Number of subscriptions authorized

Name	Type	Read/Write	Description
subscriptionsDenied	int	read	Number of subscriptions denied authorization
writes	int	read	Number of writes authorized
writesDenied	int	read	Number of writes denied authorization

Notifications

Class Name	Description
javax.management.Notification	News that a client interaction with a topic was not allowed (deprecated)

BaseThreadPoolMBean

Management interface to the basic thread pool

Attributes

Name	Type	Read/Write	Description
activeCount	int	read	The number of active threads
coreSize	int	read-write	The number of threads to maintain
keepAlive	long	read-write	Keep-alive time in milliseconds
largestSize	int	read	The largest number of threads that have simultaneously been in the pool
maximumSize	int	read-write	Maximum pool size
queueLowerThreshold	int	read	The lower queue size at which the notification handler will be notified
queueMaximumSize	int	read	The maximum queue size that the task queue can reach
queueSize	int	read	The size of the embedded task queue
queueUpperThreshold	int	read	The upper queue size at which the notification handler will be notified
size	int	read	The current number of threads in the pool
taskCount	long	read	The approximate total number of tasks that have ever been scheduled for execution

ClientConnectionStatisticsMXBean

Monitoring interface to the client session statistics MBean

Attributes

Name	Type	Read/Write	Description
clientOutputFrequency	long	read-write	Statistics output frequency in milliseconds
clientResetFrequency	long	read-write	The frequency at which the counters are reset
concurrentClientCount	int	read	The current client session count
connectionCounts	Map	read	The current client session count, broken down by client type
maximumConcurrentClientCount	int	read	The maximum number of concurrent client sessions
maximumDailyClientCount	int	read	The count of client sessions started in a day

ConnectorManageableMBean

Management interface to a connector

Attributes

Name	Type	Read/Write	Description
keepAliveQueueMaximumDepth	int	read-write	The maximum queue depth used for clients in the keep-alive state
keepAliveTime	long	read-write	The time in milliseconds that a unexpectedly disconnected client is kept alive before closing
numberOfAcceptors	int	read	The number of acceptors
queueDefinition	String	read-write	The queue definition
totalNumberOfConnections	long	read	The number of connections accepted since the connector was started
uptime	String	read	The time this connector has been running as a formatted string, or 0 if the connector is not running
uptimeMillis	long	read	The time this connector has been running in milliseconds, or 0 if the connector is not running

Operations

Name	Return Type	Arguments	Impact	Description
remove	void	0	ACTION	Remove the connector. It will not be possible to restart the

Name	Return Type	Arguments	Impact	Description
				connector again (until system restart).

Name	Return Type	Arguments	Impact	Description
start	void	0	ACTION	Start the connector

Name	Return Type	Arguments	Impact	Description
stop	void	0	ACTION	Stop the connector. Allows it to be restarted.

DiffusionControllerMBean

Diffusion Controller

Attributes

Name	Type	Read/Write	Description
buildDate	String	read	The build date and time
freeMemory	long	read	The amount of free memory in the Java virtual machine
licenseExpiryDate	Date	read	The license expiry date
maxMemory	long	read	The total amount of memory in the Java virtual machine
numberOfTopics	long	read	The number of topics on this server
release	String	read	The Diffusion release string, for example, 5.1.2_01
startDate	Date	read	The date and time at which this Diffusion server was started
startDateMillis	long	read	The time at which this Diffusion server was started, as milliseconds since the epoch
timeZone	String	read	The name of the time zone to which this Diffusion server belongs
totalMemory	long	read	The total amount of memory in the Java virtual machine
uptime	String	read	The time that this controller has been running, as a formatted string. For example, "3 hours 4 minutes 23 seconds"
uptimeMillis	long	read	The time this controller has been running, in milliseconds
usedPhysicalMemorySize	long	read	Free physical memory, in bytes
usedSwapSpaceSize	long	read	Used swap space, in bytes

Name	Type	Read/Write	Description
userDirectory	String	read	The directory in which the Diffusion server was started
userName	String	read	The name of the user to whom the Diffusion server belongs

Operations

Name	Return Type	Arguments	Impact	Description
checkLicense	void	0	ACTION	Recheck the license being used

Name	Return Type	Arguments	Impact	Description
stopController	void	0	ACTION	Stop this Diffusion controller

Name	Return Type	Arguments	Impact	Description
stopController	void	2	ACTION	Stop this Diffusion controller, and record the reason and adminName

Argument name	Type	Description
reason	String	The reason this server is stopping
adminName	String	The name of the administrator

JMXAdapterMXBean

Management interface to the adapter that reflects MBean attributes and notifications as Diffusion topics

Attributes

Name	Type	Read/Write	Description
updateFrequency	long	read	MBean attribute poll frequency, in milliseconds

LogFileMXBean

Management interface for Log Definition

Attributes

Name	Type	Read/Write	Description
description	LogDescription	read	The LogDescription for this log
filename	String	read	The fully qualified filename of this log
logLevel	String	read-write	The current log level as a string

MultiplexerMBean

Management interface to a multiplexer

Attributes

Name	Type	Read/Write	Description
latencyWarningTime	long	read	The latency threshold of this multiplexer, after which notifications will be sent
pendingEvents	int	read	The number of operations pending on the multiplexer queue
name	String	read	The Multiplexer name
numberOfClients	int	read	The current number of clients assigned to multiplexer

Notifications

Class Name	Description
javax.management.Notification	Published in case of multiplexer latency

MultiplexerManagerMBean

Management interface to the multiplexer manager

Attributes

Name	Type	Read/Write	Description
numberOfMultiplexers	int	read	The number of multiplexers

PublisherControllerMBean

Management interface for a publisher

Attributes

Name	Type	Read/Write	Description
inboundClientAverageMessageSize	long	read	The average size of a message received from clients
inboundClientNumberOfMessages	long	read	The count of messages received from clients
logLevel	String	read-write	The log level for this publisher
numberOfTopics	int	read	The count of topics associated with this publisher
outboundAverageMessageSize	long	read	The average size of a message sent to clients
outboundNumberOfMessages	long	read	The count of messages sent to clients
started	boolean	read	True if the publisher is started

Operations

Name	Return Type	Arguments	Impact	Description
removePublisher	void	0	ACTION	Permanently remove the publisher

Name	Return Type	Arguments	Impact	Description
startPublisher	void	0	ACTION	Start this publisher

Name	Return Type	Arguments	Impact	Description
stopPublisher	void	0	ACTION	Stop this publisher

Name	Return Type	Arguments	Impact	Description
undeploy	void	0	ACTION	Undeploy this publisher

StatisticsControllerMBean

Interface for StatisticsService controller

Attributes

Name	Type	Read/Write	Description
clientStatisticsEnabled	boolean	read-write	True if aggregate statistics for clients are enabled
enabled	boolean	read-write	True if the statistics service is enabled
publisherStatisticsEnabled	boolean	read-write	True if aggregate statistics for publishers are enabled
topicStatisticsEnabled	boolean	read-write	True if aggregate statistics for topics are enabled

StatisticsManagerMBean

Monitoring interface to the collection of server statistics

Attributes

Name	Type	Read/Write	Description
statisticsEnabled	boolean	read-write	True if global statistics are enabled

VirtualHostMBean

HTTP virtual host management interface

Attributes

Name	Type	Read/Write	Description
cacheSizeBytes	int	read	The cache size in bytes
cacheSizeEntries	int	read	The number of entries in the cache

Name	Type	Read/Write	Description
aliasFile	String	read	the alias file name
aliasProcessor	HTTPAliasProcessor	read	the alias processor object
cache	HTTPCache	read	the HTTP cache
compressionThreshold	int	read	the compression threshold
debug	boolean	read	true if debug is set
documentRoot	String	read	the document root directory
errorPage	String	read	the error-page file name
fileServiceName	String	read	the file service name
homePage	String	read	the home-page file name
host	String	read	the host name
minify	boolean	read	true if the minify property is set
name	String	read	the virtual host name
numberOfRequests	int	read	number of requests actioned since service was started
static	boolean	read	true if static
webServerName	String	read	the web server name

Operations

Name	Return Type	Arguments	Impact	Description
startService	void	0	ACTION	Restart a previously stopped virtual host

Name	Return Type	Arguments	Impact	Description
stopService	void	0	ACTION	Stops the virtual host from processing requests

Name	Return Type	Arguments	Impact	Description
clearCache	void	0	ACTION	Clear the cache of all entries

Statistics

Diffusion provides statistics about the server, publishers, clients, and topics.

Statistics exposed by MBeans

Diffusion provides a set of statistics as MBeans. For more information about these statistics, see MBeans on page 676.

These statistics can be accessed in the following ways:

- Using a JMX tool, such as VisualVM or JConsole. For more information, see Using Java VisualVM on page 671 or Using JConsole on page 673.
- Using the Diffusion monitoring console. For more information, see Diffusion monitoring console on page 688.
- Through topics under the topic `Diffusion/Metrics`.

You can configure whether these statistics are collected and how they are reported. For more information, see Configuring statistics on page 687.

You can configure Diffusion to report the top-level statistics through JMX. These statistics appear under `com.pushtechnology.diffusion.metrics`

The following is a list of all the top level statistics and their attributes.

- clients.concurrent
  - Count
- clients.concurrent_max
  - Value
- client.connections
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate
- clients.disconnections
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate
- eventpublishers.concurrent
  - Count
- eventpublishers.connections
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate
- eventpublishers.disconnections
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate
- publishers.loaded
  - Count

- publishers.started
  - Count
- topics.additions
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate
- topics.count
  - Count
- topics.deletions
  - RateUnit
  - Count
  - FifteenMinuteRate
  - FiveMinuteRate
  - MeanRate
  - OneMinuteRate

In all top-level statistics that attributes have the following values:

Count

The total number of items or events of this type over the lifetime of the metric.

RateUnit

The unit for the various rates. Generally, this is events per second.

MeanRate

The count divided by the lifetime of the metric.

OneMinuteRate, FiveMinuteRate, FifteenMinuteRate

A moving average across the given time period that is exponentially weighted towards new data.

Statistics in the Publisher API

You can get some statistics from the Publisher API. For more information, see the Java Unified API documentation.

The following statistics are provided through the Publisher API:

- PublisherStatistics
  - InboundClientMessageStatistics
    - AverageMessageSize
    - BytesOnWire
    - NumberOfMessages
  - OutboundMessageStatistics
    - AverageMessageSize
    - BytesOnWire
    - NumberOfMessages
- ClientStatistics
  - InboundMessageStatistics

        - AverageMessageSize
        - BytesOnWire
        - NumberOfMessages
    - OutboundMessageStatistics
        - AverageMessageSize
        - BytesOnWire
        - NumberOfMessages
- TopicStatistics
    - InboundMessageStatistics
        - AverageMessageSize
        - BytesOnWire
        - NumberOfMessages
    - OutboundMessageStatistics
        - AverageMessageSize
        - BytesOnWire
        - NumberOfMessages
    - TotalNumberOfSubscribers

Configuring statistics

You can configure statistics using the `etc/Statistics.xml` configuration file or programmatically using the Classic API.

The `Statistics.xml` configuration file

Diffusion servers provide statistics which are available through the JMX MBeans or through topics under the topic `Diffusion/Metrics`. If statistics is enabled via `etc/Statistics.xml`, users can take measurements including the average message size, number of messages per topic per second, etc.

The statistics configuration has several distinct elements, that allow granular control over what is enabled on server start.

- `<statistics>`

  The top-level element. Setting the enabled property to false will disable all statistics for the server.
- `<topics>`, `<clients>`, `<publishers>`, and `<server>`

  Enabling these elements provides aggregate metrics for each given class. Each element also contains a `<monitor-instances>` element that dictates whether specific instances of the parent class are monitored. Instance metrics require that the parent element is enabled.

The collection of metrics is configured separately from the distribution thereof. Within the `Statistics.xml` configuration file, there is also a `<reporters>` element which contains definitions of available reporters, which expose metrics over different mediums.

Some reporters allow certain properties to be passed to them. For instance, the topics reporter allows the use of `<property name="interval">` to provide the desired update frequency in seconds. Details of valid properties is documented within the `etc/Statistics.xml` file itself.

Configuring a reporter to distribute statistics through JMX

Inside the `reporters` element of the `Statistics.xml` configuration file, add the following XML:

```
<reporter name="JMX" enabled="true">
    <type>JMX</type>
</reporter>
```

Configuring a reporter to distribute statistics through topics

Inside the `reporters` element of the `Statistics.xml` configuration file, add the following XML:

```
<reporter name="Topics" enabled="true">
    <type>TOPIC</type>
</reporter>
```

Programmatic configuration

You can programmatically query and configure the recording and calculation of statistics for the classes:

- `com.pushtechnology.diffusion.api.publisher.Client`
- `com.pushtechnology.diffusion.api.topic.Topic`
- `com.pushtechnology.diffusion.api.publisher.Publisher`

Developers are able to query the state of each class, to discover whether it is recording statistics using method `isStatisticsEnabled()`, stop recording with `stopStatistics()` and start recording using `startStatistics()`. The relevant API documentation holds more detail on the subject.

It has been observed that in a system with significant numbers of clients enabling the client instance statistics can result in a performance impact of up to 20% reduction in maximum throughput achieved, which can inhibit the system from supporting further connections.

If your system has more than 20k clients (at any given time) per Diffusion instance we recommend the client instance statistics be turned off.

Similarly if your system supports very large number of topics turning on the topic instance statistics might result in a performance hit.

By default statistic collection is turned off for performance reasons, to enable statistic collection, set the statistics root attribute `enabled`in `Log.xml` to true.

Diffusion monitoring console

A web console for monitoring the Diffusion server.

About

The Diffusion Monitoring Console is an optional publisher, provided as `console.dar`. It is deployed by default, and can be undeployed in the same manner as any DAR file. It exists to give operational staff using a web browser accessible visibility over the operations of a Diffusion solution

To manage a Diffusion server and make changes to it, use JMX tools such as JConsole. Unless you have to stop the Diffusion server, and stop and restart a publisher.

Dependencies

The console depends on the Diffusion publisher to mirror JMX MBeans as topics. The console also makes use of the statistics controlled by `etc/Statistics.xml`

The live graphing feature mandates a web browser that supports Scalar Vector Graphics (SVG). Most modern web browsers implement the features required by the Console however Internet Explorer v9 is the recommended minimum for Microsoft users.

There are also two configuration settings within the Diffusion publisher configuration within `etc/Publishers.xml`. These are:

- `console.control.server` – Enable the ability to stop the Diffusion server through the console.
- `console.control.publishers` – Enable the ability to stop a particular publisher through the console.

Both of these options are disabled by default.

Logging in

The console is secured by a username and password. The username you use to login must have permissions to view and act on information on the Diffusion server.

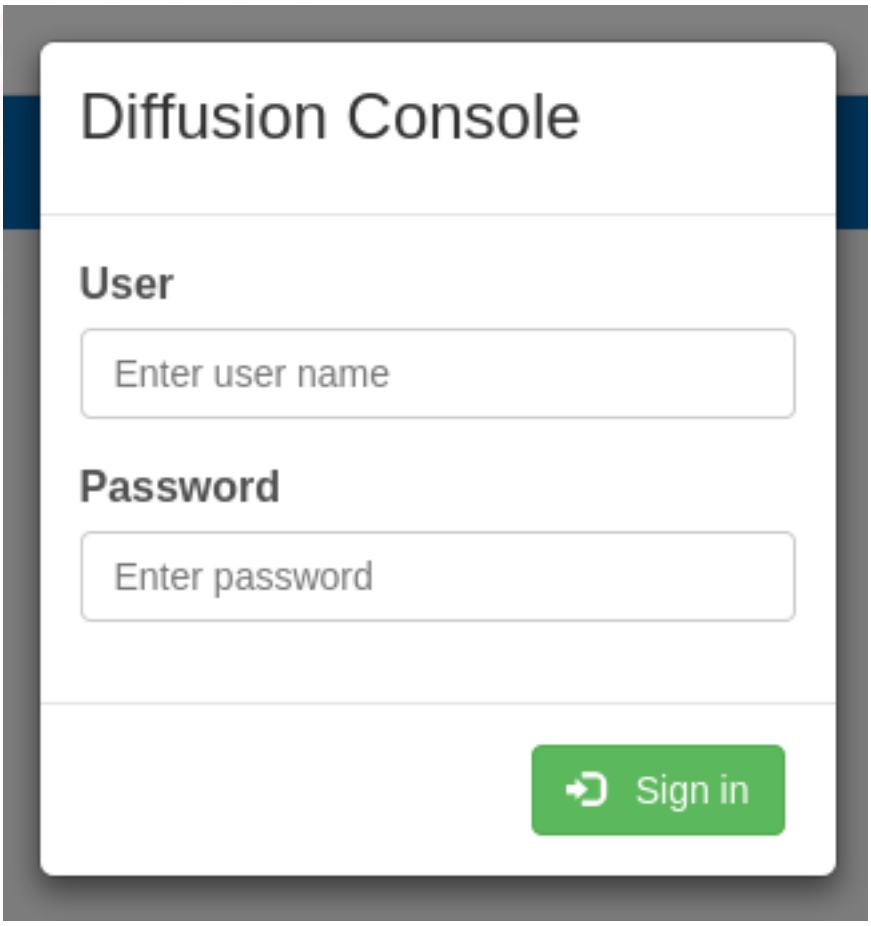


Figure 66: Logging in the monitoring console

Features: Dashboard

Dashboard panels

By default the console is deployed as part of Diffusion. It is available in a fresh installation at http://localhost:8080/console.

Default layout

By default, the console consists of six panels, each focusing on a key feature of the server.

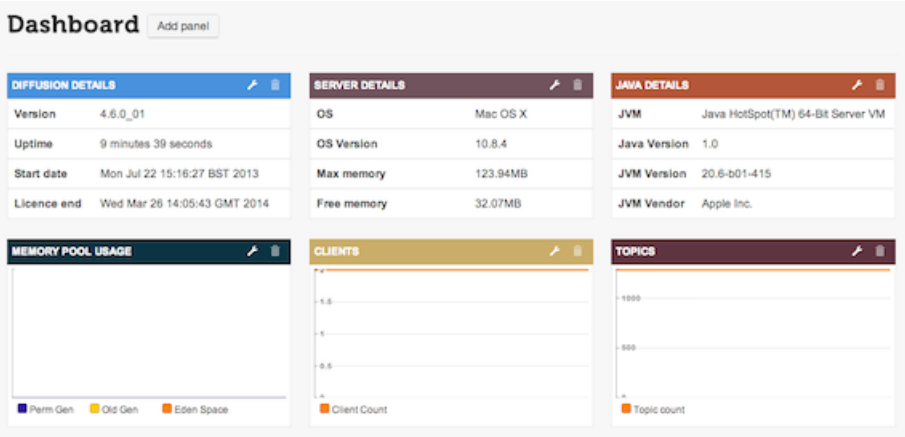


Figure 67: The default console layout

- **Diffusion Details**: the server version; the server up time, the server start date and time and the time and which the current license expires.
- **Server details**: the name and version of the underlying operating system; the total memory available (physical and virtual) and the amount of free memory.
- **Java Details**: the name, vendor and version of the JVM.

Instead of tabular data the second row show live line graphs.

- **Memory pool usage**: the values over time of the memory used by the JVM process.
- **Clients**: the value over time of the number of clients connected.
- **Topics**: the value over time of the number of topics on your Diffusion server.

Publishers table

At the bottom of the Dashboard is the publishers table. At a glance this shows the installed publishers and their vital statistics: the number of topics created, client connected, messages sent, bytes sent and finally publisher status

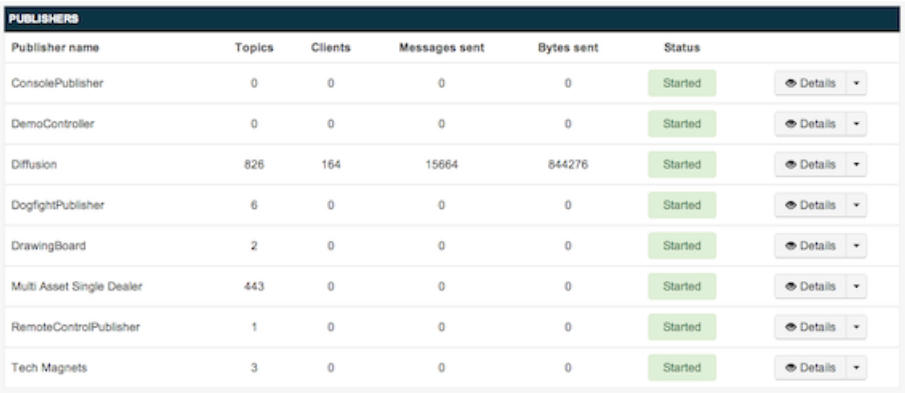

PUBLISHERS
Publisher name Topics Clients Messages sent Bytes sent Status
ConsolePublisher 0 0 0 0 Started Details
DemoController 0 0 0 0 Started Details
Diffusion 826 164 15664 844276 Started Details
DogfightPublisher 6 0 0 0 Started Details
DrawingBoard 2 0 0 0 Started Details
Multi Asset Single Dealer 443 0 0 0 Started Details
RemoteControlPublisher 1 0 0 0 Started Details
Tech Magnets 3 0 0 0 Started Details

Figure 68: The table of publishers

Using the pull-down menu on the **Details** button publishers can be stopped and restarted. The **Details** button itself reveals the publisher statistics: clients, topics, average messages per second and average bytes per second.

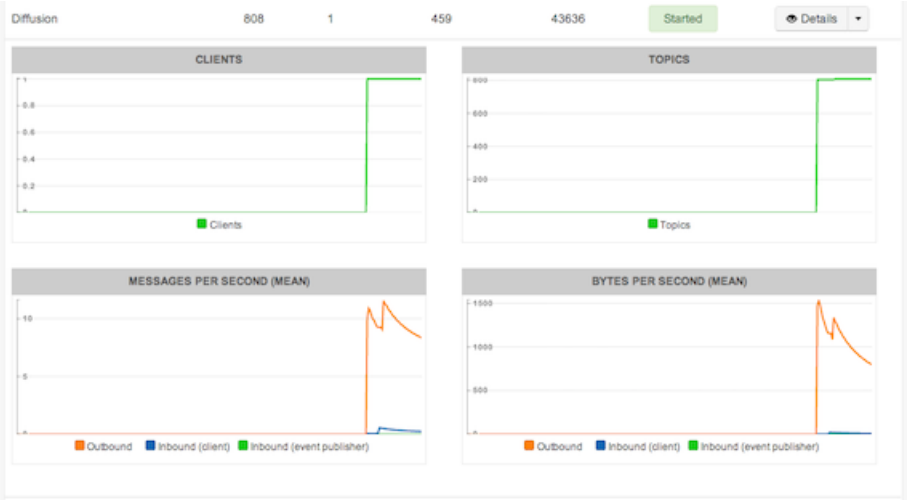


Figure 69: Publisher statistics graphs

Features: Topics tab

The Topics tab brings to the web browser the ability to browse and interact with the Diffusion topic tree.

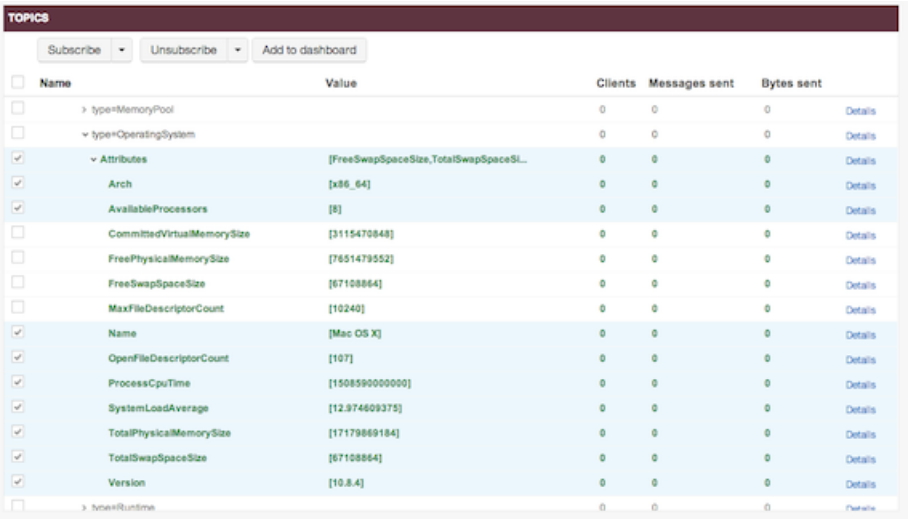


Figure 70: The table of topics

Users can intuitively browse the live topic tree, fetch and subscribe to topics. If the server is so configured the table also shows the number of subscribed clients, messages sent and bytes sent. Enable individual topic statistics through `etc/Statistics.xml`, for example,

```
<!-- Enable global topic statistics -->
  <topic-statistics enabled="true">
  <!-- Enable individual topic instance statistics -->
  <monitor-instances>true</monitor-instances>
</topic-statistics>
```

Once a set of topics is selected using its checkbox the Subscribe and Unsubscribe work intuitively, and each button has an recursive alternative available through the drop-down menu-button.

The details button shows more detail on the topic in question, as well as offering to fetch the topic value

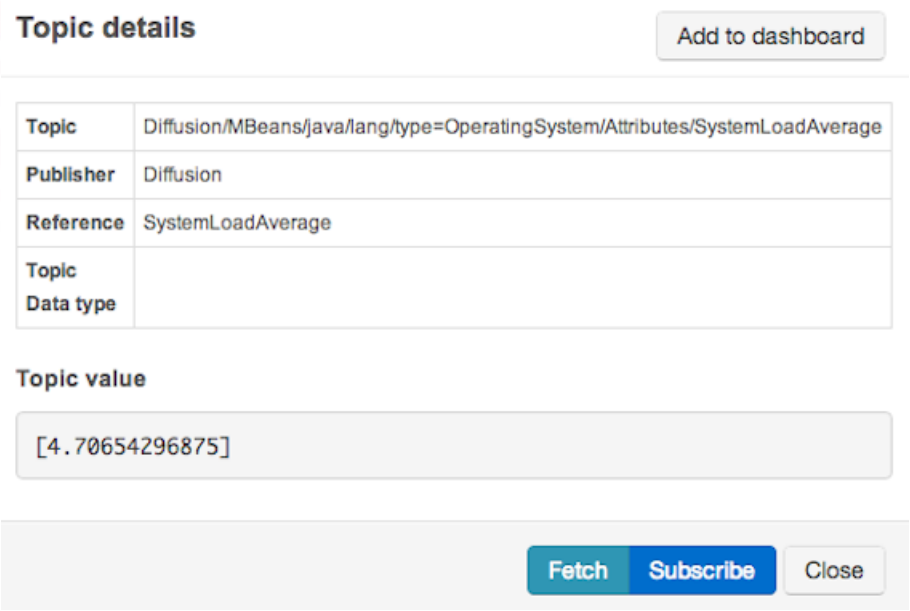


Figure 71: Details of the topic publishing the CPU load of the host server

Features: Clients tab

The Client tab shows a live list of the clients connected to the Diffusion server. Additionally it shows the number of messages to and from the server, the client IP address, connection type and connection time.

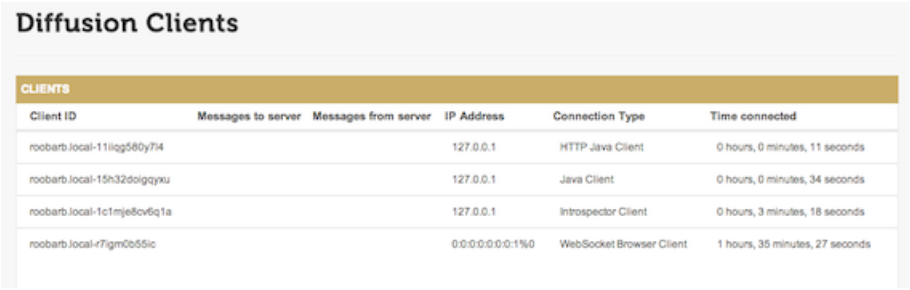


Figure 72: The table of clients

Configure the Diffusion server to provide live client statistics through `etc/Statistics.xml`

```
<!-- Enable global client statistics -->
<client-statistics enabled="true">
  <!-- Definition of the log in Logs.xml -->
  <log-name>stats</log-name>
  <!-- Specifies the output frequency of the log, this is one
 entry per frequency -->

  <output-frequency>1h</output-frequency>
  <!-- Enable individual client instance statistics -->
  <monitor-instances>true</monitor-instances>
```

```
</client-statistics>
```

Features: Logs tab

The Logs tab shows a live color-coded display of log entries emitted by the server at the levels of **INFO**, **WARN**, and **ERROR**.

Server logs

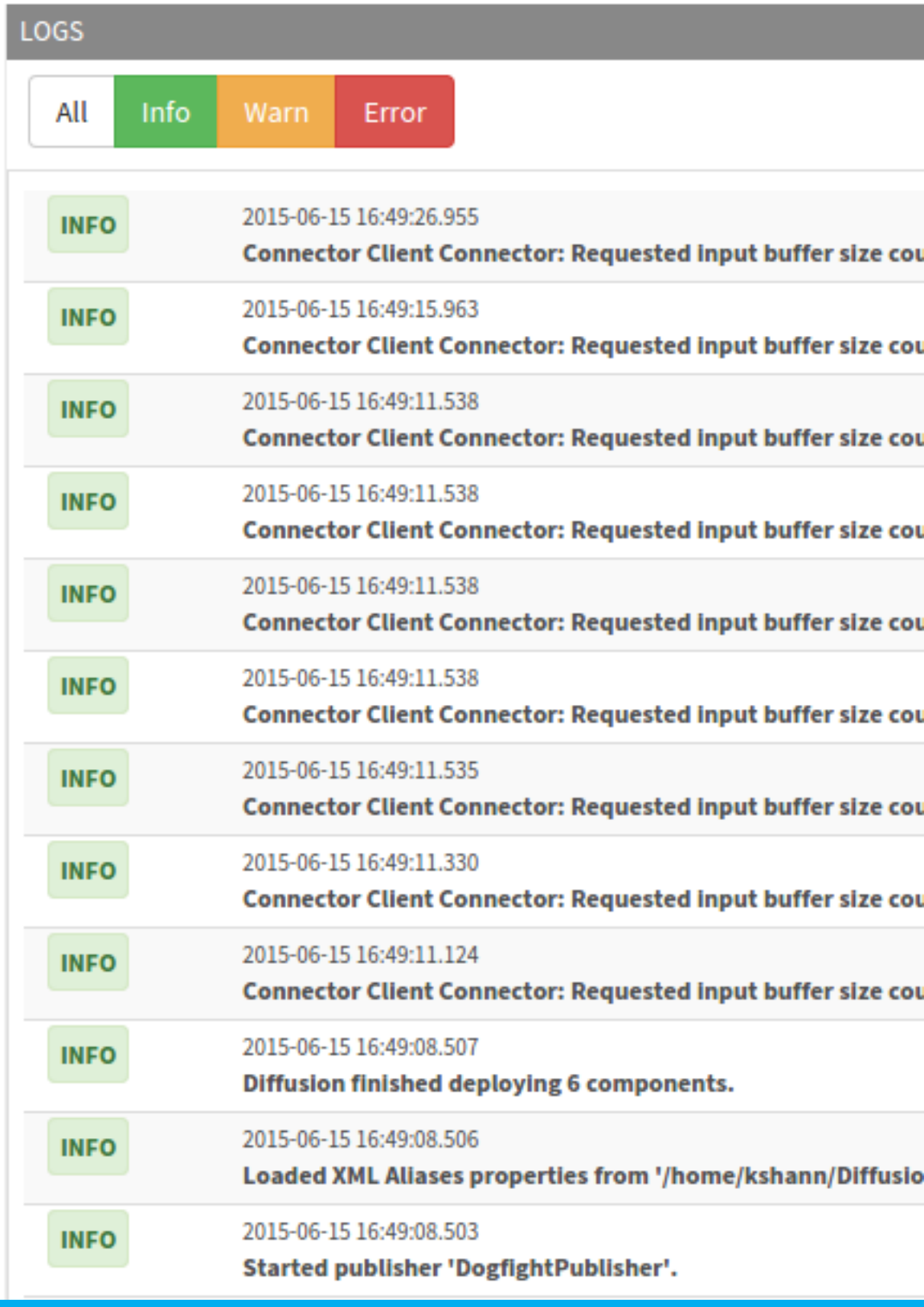


Figure 73: The table of log entries

Users can also perform client-side simple filtering on log entries. Unlike other monitoring metrics the Diffusion server retains up to 250 log entries in memory.

Advanced

Saving the console layout

Users can save changes made to their console layout with the “Save Dashboard layout” button. This persists a file on the server side, making it shared amongst all Console users.

White & Blacklist editing

The Console optionally maintains a blacklist or whitelist of IP addresses that are allowed to make use of it. Users can specify discrete IP addresses or use syntax supported by `etc/SubscriptionValidationPolicy.xml` to cover subnets. In order to make these changes active, after editing the whitelist or blacklist and clicking the "Save settings" button, you must restart the server.

Figure 74: Editing the Access Policy

Stop Diffusion

Required permissions: control_server

The **Stop Diffusion** menu item stops the server when clicked upon.

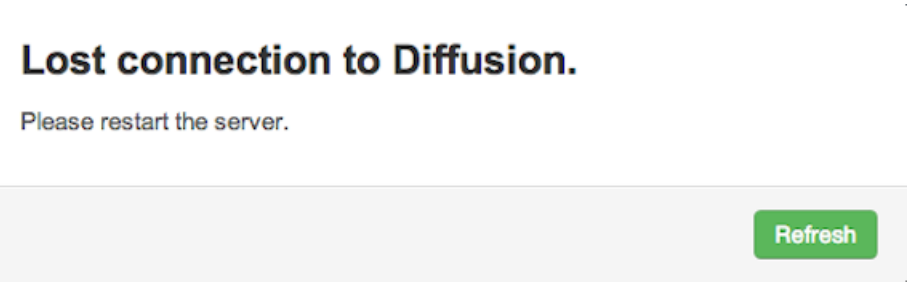


Figure 75: Notification that the Diffusion server has stopped

Going further

Changing the console layout

The console is designed to be extensible and flexible. Users can reorder, edit, create and remove panels. Grab the panel header and drag it to a new location as desired. Click the trash-can icon to remove the icon – with verification

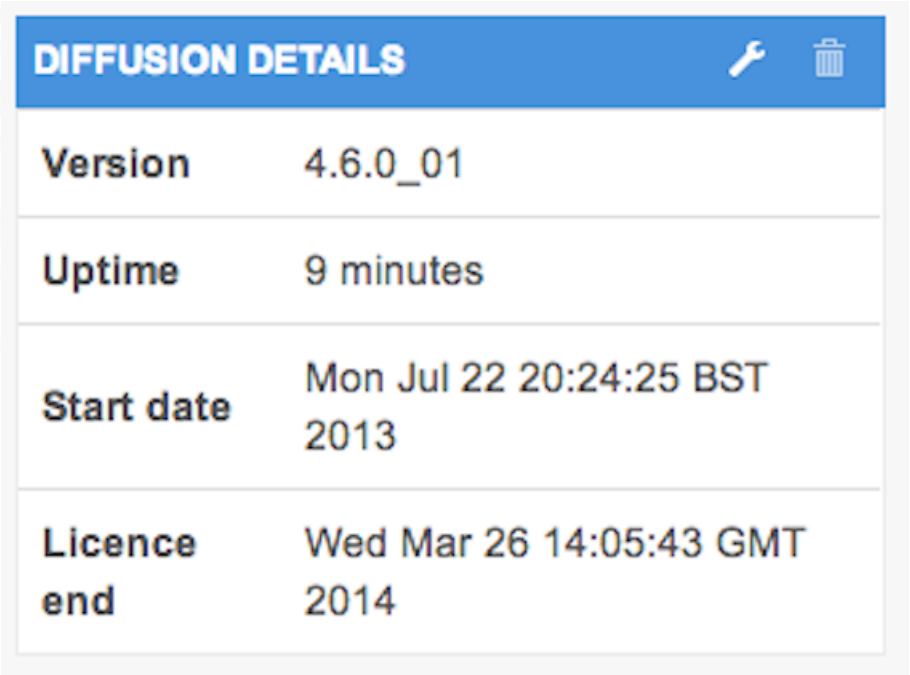


Figure 76: The default Diffusion Details panel

Click on the spanner or wrench icon to configure the panel.

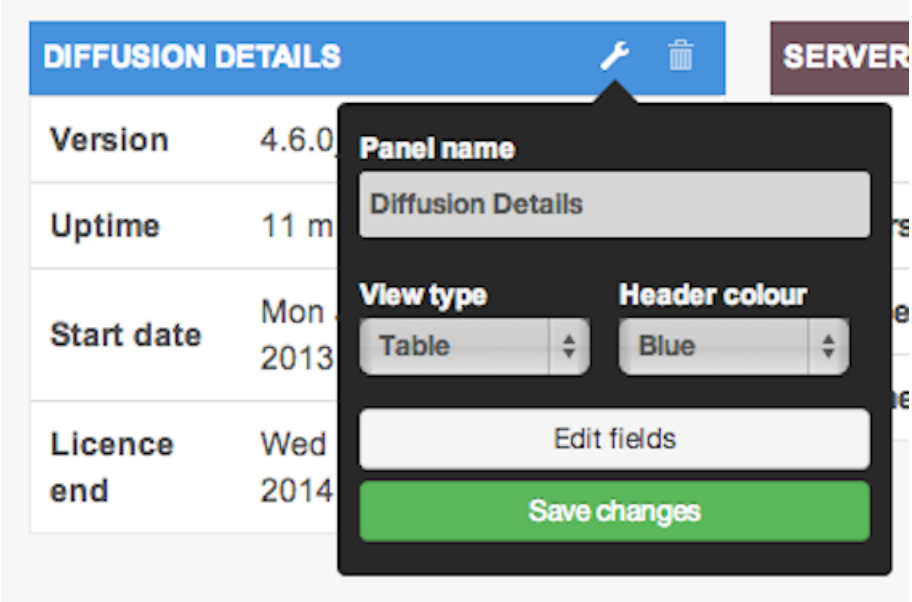


Figure 77: Editing the properties of the Diffusion Details panel

Panel name' and 'Header color' are self explanatory. 'View type' is a choice of data renderings.

- **Table**: As seen already, this option shows one or more monitoring metrics in a table of textual values.
- **Line**: Renders one or more numeric metrics as a line graph.
- **Area**: Renders one or more numeric metrics as an overlapping area graph.
- **Single**: Used to visualize a single metric in large text, for metrics that are worth the screen real-estate.

Line and area graphs have an extra two configuration fields: 'Refresh rate (ms)' and 'Max data points'. The latter configures how much data is retained for rendering the graph. The former governs the frequency with which the graph is updated and does not influence the frequency of updates from the server. Historic data is only stored in the browser and refreshing the page loses the stored set of data points.

Hovering the mouse over a graph panel shows the detail of the underlying data point

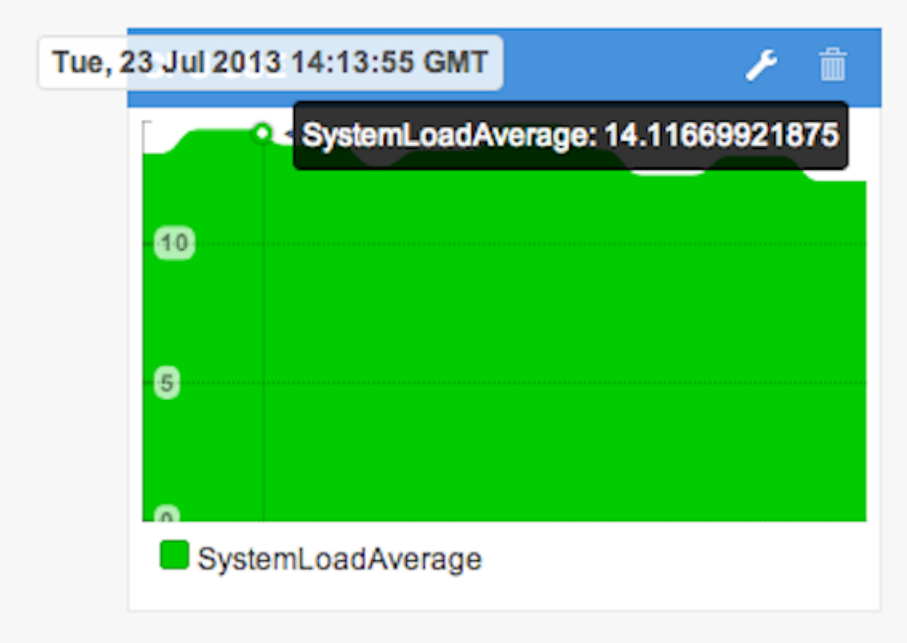


Figure 78: Visualizing the CPU load on a server at a specific time.

Sourcing monitoring metrics

Clicking the 'Edit fields' button presents the user with a Topic Data Fields dialog, where the user nominates one or more topics from where metrics are drawn.

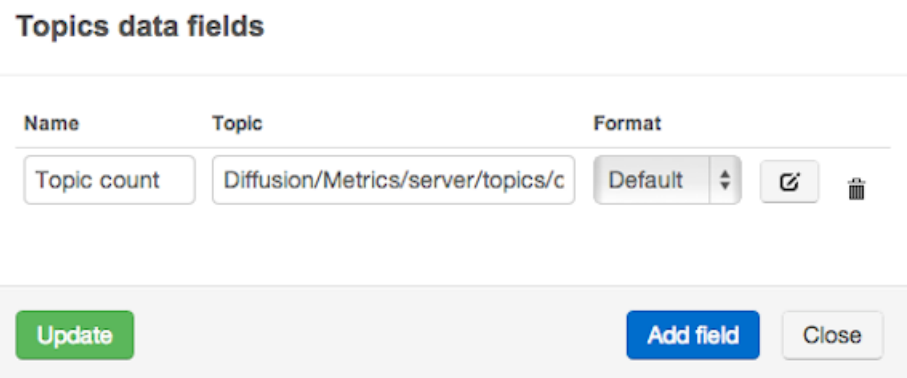


Figure 79: Editing and adding to the set of topics for this panel

Users of the **Topics** tab have already seen the **Add to dashboard** button in the **Topic details** dialog that can shortcut this process.

The default Console layout draws metrics from topics in the Diffusion/MBeans topic tree, however this is not mandatory and solution implementers are free to draw on any suitable topic to reflect their own monitoring needs – including 3rd party topics implemented as part of the solution.

The `Diffusion/MBeans` topic tree is populated by the JMX adapter which reflects all JMX MBeans as topics. Solution implementers that build custom MBeans to manage their solution can re-use the same MBeans for monitoring purposes.

The Console can draw on features that are themselves optional (Topic and client statistics, for example). If they are disabled, the Console points this out, and request they be enabled in `etc/Statistics.xml`

Production deployment notes

Securing the `Diffusion/` topics

The topics in the `Diffusion/` tree convey a great deal of power and it is highly probable that bringing a Diffusion based solution to production requires limiting their access to suitable users: for example, users with an IP address in a specific range. Solution implementers can achieve this by implementing an authentication handler.

The default configuration for the console allows users to stop and restart publishers as well as stop the Diffusion server itself. This feature is configured using the properties `console.control.server` and `console.control.publishers` on the Diffusion publisher in `etc/Publishers.xml`.

Basic integration with Splunk

How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

About

Splunk is a third-party application from Splunk, Inc., which provides monitoring and analysis of other applications, primarily by parsing their logs and extracting information of interest. The information is displayed through a web interface, which allows the creation of dashboards and alerts on user-defined events. Splunk is available for all major operating systems.

The Diffusion log format is designed to be consistent and to allow for easy parsing by monitoring tools, not limited to Splunk.

Installation

Installation typically takes just a few minutes, see the appropriate section of the Splunk Installation Manual. For simplicity, we assume that Diffusion and Splunk are installed on the same machine.

Basic configuration

This is easier to do with existing log files to import, so configure Diffusion to write log files. To better demonstrate Splunk, set the server log file to TRACE logging in `etc/Logs.xml` and start Diffusion.

```
<!-- Example server log configuration -->
<log name="server">
 <log-directory>../logs</log-directory>
 <file-pattern>%s.log</file-pattern>
 <level>TRACE</level>
 <xml-format>false</xml-format>
 <file-limit>0</file-limit>
 <file-append>false</file-append>
 <file-count>1</file-count>
 <rotate-daily>false</rotate-daily>
```

```
</log>
```

On startup, access the Splunk web UI at http://localhost:8000. After logging in (and changing the default admin password), choose the **Add data** option.

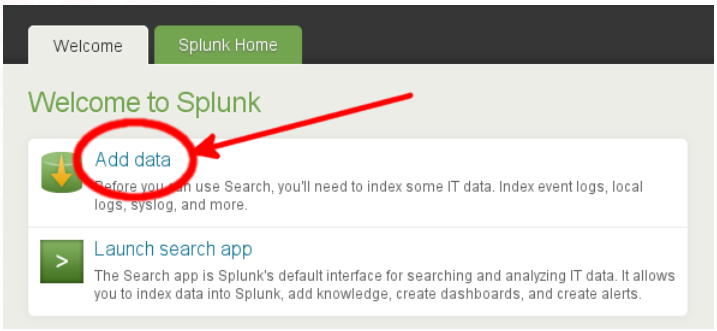
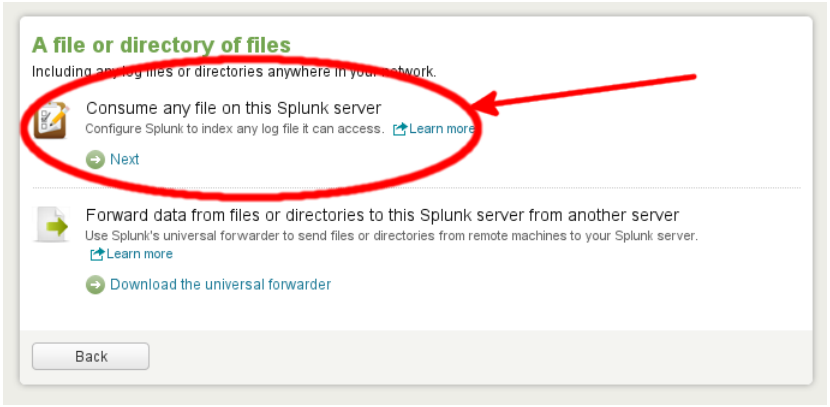

Welcome
Splunk Home
Welcome to Splunk
Add data
Launch search app
The Search app is Splunk's default interface for searching and analyzing IT data. It allows you to index data into Splunk, add knowledge, create dashboards, and create alerts.

Figure 80: Welcome tab of the Splunk web UI

In the **Add Data to Splunk** screen that follows, choose the link **A file or directory of files** followed by **Consume any file on this Splunk server**.


A file or directory of files
Consume any file on this Splunk server
Configure Splunk to index any log file it can access.
Learn more
Next
Forward data from files or directories to this Splunk server from another server
Use Splunk's universal forwarder to send files or directories from remote machines to your Splunk server.
Learn more
Download the universal forwarder
Back

Splunk might not be able to immediately identify the format of the log files; if this is the case, a dialog box similar to the following is presented. Select **csv** from the existing source types. Diffusion uses a pipe symbol rather than a comma as a separator but this is acceptable to the Splunk CSV parser.

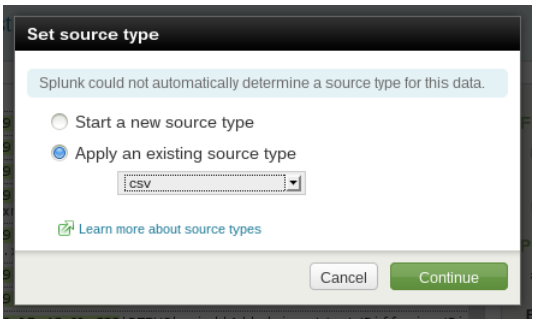

Set source type
Splunk could not automatically determine a source type for this data.
Start a new source type
Apply an existing source type
csv
Learn more about source types
Cancel
Continue

Figure 81: The Splunk Set source type dialog

The next dialog allows you to select the Diffusion `logs/Server.log` file under the **Preview data before indexing** option, which Splunk reads and parses. On the **Data Preview** screen, there are numbered log entries with the timestamp highlighted. This indicates that the log file has been correctly parsed. Accept this, and on the next screen, set the source to be continuously indexing the data. You can leave the parameters in **More settings** at their default values. Once

this is done, you have given the new data source a name (for example, Diffusion Server Log) and finally accepted the settings, you can begin searching and generating reports based on the log contents.

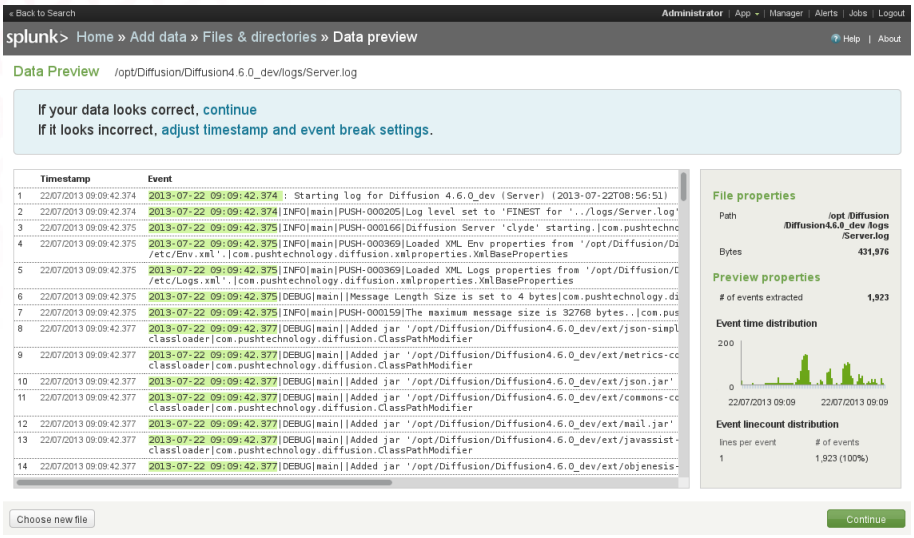


Figure 82: The Data Preview panel

Simple searches

Now we have a data source configured, we can start to execute basic searches.

On the Splunk launch page, select the **Search** option. On the **Search Summary** page that opens, select the Source relating to the file `logs/Server.log` previously imported. The page changes to include the source in the Search area. Additional search terms can be added to the end, for example, “Started Publisher”.

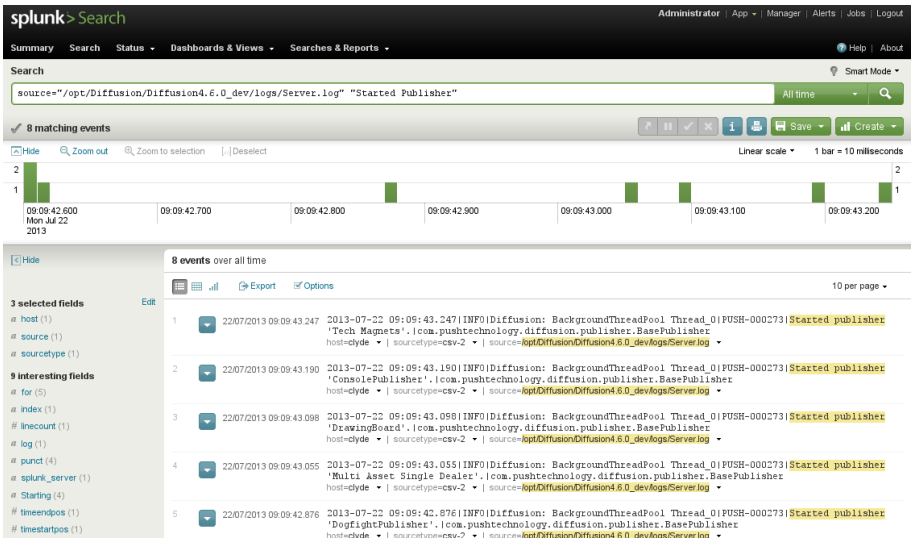


Figure 83: The Splunk search summary panel

Related Links

Logging on page 721
The Diffusion server logs messages about its activities and any problems that occur. Log messages are written to the console (the process output, which can appear in a terminal window) and to a log file. The level of logging can be adjusted, from a moderate rate of

messages under normal circumstances for system monitoring, to very detailed logging for problem diagnosis.

http://docs.splunk.com

Network security

This section describes how to deploy network security, which can be used in conjunction with data security.

Secure clients

Diffusion clients can connect to your solution using TLS, commonly called SSL. The TLS can terminate at your load balancer or at your Diffusion server. Terminating the TLS at the load balancer reduces CPU cost on your Diffusion servers.

Below is a table of the clients and whether they can connect securely through SSL and/or HTTPS.

Table 82: Client security

Client	DPTS	HTTPS
.NET	✓	✗
Java	✓	✓
C	✓	✗
Flash	✓	✓
Silverlight	✗	✓
WebSocket	✓	✗
XHR	✗	✓
Android	✓	✗
iOS	✓	✗

You can have Diffusion server primary/replica connections over DPTS as well.

Cipher suite configuration

By default, Diffusion supports only those SSL cipher suites with no known vulnerabilities:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA256

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA

The following protocols are supported by default:

- SSLv2Hello
- TLSv1
- TLSv1.1
- TLSv1.2

You can use the system property `https.cipherSuites` when starting the Diffusion server to override the supported set of cipher suites.

```
-Dhttps.cipherSuites={"TLS_RSA_WITH_AES_128_CBC_SHA",
 "TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA"}
```

Web server configuration

The web server can be configured in your test environment to allow you to deploy and undeploy DAR files by using a web service. By default this capability is not enabled.

For security, if you choose to enable this web service in your production environment, you must restrict access to the *`diffusion-url`*`/deploy` URL by other means. For example, by setting up restrictions in your firewall.

To configure the web server, use the `WebServer.xml` file. For more information, see WebServer.xml on page 642. An example of this file is provided in the `/etc` directory of the Diffusion installation. The XSD is provided in the `/xsd` directory of the Diffusion installation.

Connector configuration

If secure connections are required, Diffusion connectors must be configured to support DPTS and/or HTTPS. Any connector can accept DPTS connections. A connector does not have to be dedicated to only DPTS connections. DPTS connections can also accept clear-text connections. To enable DPTS connections a keystore entry is required in the connector configuration. This informs the connector that it is enabled for DPTS type connections. If HTTPS is required, a keystore section and a web-server entry are also required, even for secure Diffusion clients.

To configure the connectors, use the `Connectors.xml` file. For more information, see Connectors.xml on page 607. An example of this file is provided in the `/etc` directory of the Diffusion installation. The XSD is provided in the `/xsd` directory of the Diffusion installation.

Keystores

The default Diffusion installation includes a sample keystore containing a self-signed certificate. This is suitable for development. The certificate will not be trusted by browsers and other clients without additional configuration. If you use TLS in production, you must create a new keystore, using a certificate obtained from a certificate authority.

The following steps use the Java Keytool to create a keystore. The steps can vary depending on your certificate authority. For more information, refer to your certificate authority's documentation.

1. Generate a key and place it in your keystore.

   ```
   keytool -genkeypair -alias my_alias -keyalg RSA -
   keystore keystore_name -keysize
   ```

2. Generate a CSR file.

```
keytool -certreq -keyalg RSA -alias my_alias -file certreq.csr
 -keystore keystore_name
```

3. Send the CSR file to your certificate authority.
4. Receive the signed certificate from your certificate authority.
5. Install any intermediate certificates that you require.

```
keytool -import -trustcacerts -alias intermediate_alias -
keystore keystore_name -file intermediate_certificate_file.crt
```

6. Install your own certificate. Use the same alias as when you generated the key and the signing request.

```
keytool -import -trustcacerts -alias my_alias -
keystore keystore_name -file certificate_file.crt
```

Provided keystores

The `etc` directory of the Diffusion directory contains the following keystores:

sample.keystore

This keystore is an example keystore that contains a self-signed certificate. In production, we recommend you create your own keystore that contains a certificate signed by a certificate authority.

licence.keystore

This keystore contains the public key used for the Diffusion license file. Do not edit or delete this keystore. Diffusion requires this keystore to verify your Diffusion license.

Related Links

http://www.networking4all.com/en/support/ssl+certificates/manuals/java/java+based+webserver/keytool+commands/

Tuning

Aspects of tuning Diffusion for better performance or resilience

This section covers aspects of configuring Diffusion to achieve higher levels of performance and covers some of the more advanced features which enable users to get more out of Diffusion.

Buffer sizing

There are a number of places within the configuration of Diffusion where buffer sizes must be specified and getting these right can have a significant impact upon performance.

Connector output buffers

An output buffer size must be configured for each connector.

The output buffer size configured for a connector must be at least as large as the largest message that is expected to be sent to any client connecting through that connector. However, the buffer size can be much larger so that the messages can be batched at the server, which improves performance.

Each connected client is assigned a socket buffer of the specified size if possible. A warning is logged if a smaller socket buffer was allocated than requested.

In addition each client multiplexer has a buffer of the configured size (as a multiplexer writes to only one of its clients at any one time). The multiplexer buffer is used to batch messages from the client queue before writing and, if the socket buffer does end up being smaller than the configured buffer and the throughput is high, the allocated socket buffer size might become a bottleneck.

Getting the correct output buffer size is vital. Make this too small and the Diffusion server does not batch and write messages to clients at optimal rates. Make them too big, extra memory is consumed or messages might time out and cause the client connection to be closed.

If the output buffer size is larger than the TCP output buffer size, this can cause problems if the client is slow consuming. If a slow-consuming client does not clear messages from the TCP buffer fast enough, messages on the connector buffer which are waiting to move to the TCP output buffer can time out. You can avoid this problem by setting the TCP output buffers for your operating system and the connector output buffers for your Diffusion server to the same value. You can also increase your message timeout interval.

> **Note:** For maximum performance, ideally all clients configure their input buffer size to match the connector's output buffer size.

Client output buffers

As at the server, the output buffer sizes in use must be configured for a client.

In the Java client this is specified in the ServerDetails object used to make the connection. As the Java client does not buffer messages, this only has to be large enough to cater for the largest message size that is sent to the server.

Publisher client output buffers

A publisher client (a connection made from a publisher to another Diffusion server) is slightly different from a normal client in that it does queue and buffer messages for sending. It is advantageous to throughput to use a larger output buffer size.

The output buffer size is configured in the server element in `Publishers.xml` or in the ServerDetails object depending upon how the connection is being made.

Connector input buffers

Each connector also specifies an input buffer size.

Input buffers receive messages from clients. This buffer must be as large as the largest message expected. If you specify an input buffer size that is less than the maximum message size, the maximum message size is used as the input buffer size.

This size is also used to allocate a receive socket buffer for the client. The socket buffer allocated might actually be less than requested in which case a warning is logged.

For maximum performance, the size used for this buffer must match up with the output buffer size used by clients.

Client input buffers

Clients must also specify the buffer size to use for input.

In the Java client this is specified in the ServerDetails object used to make the connection (or possibly `Publishers.xml` for a publisher client connection).

Matching buffer sizes

For optimal throughput it is desirable to match the size of buffers at each end of every connection. The input buffer size used by clients ideally matches the output buffer size at the connector that they connect to. Also the output buffer size specified by clients must match the input buffer size of the connector they connect to.

> **Note:** Because publisher server connections queue and batch messages at both ends, use a separate connector for such connections such that optimal buffer sizing can be achieved.

Message batching

The size of output buffers configured for a connector can be much larger than the largest expected message size because the server uses these buffers to batch client messages which improves performance. Ideally the buffer size is a multiple of the average message size.

> **Note:** When using for HTTP clients, allow between 20 (for HTTP Duplex) and 250 (for browser) bytes extra for control information.

Each client multiplexer assigns an output buffer of each size specified by client connectors. So if there were 3 client connectors, each specifying a different output buffer size, and 2 client multiplexers, each multiplexer assigns 3 different buffers (6 in total). When using HTTP Duplex connections, each multiplexer assigns an additional buffer for chunked encoding.

When a client multiplexer is unable to write the contents of an output buffer to a client in one go, the writing is deferred and the multiplexer takes a copy of the remaining data in the output buffer into its own temporary buffer.

Message sizing

The sizing of messages that are sent to clients is very important to the overall performance and this must be carefully considered within the design of publishers.

Every topic message has a fixed header of 6 bytes. It then has the topic name terminated by one byte, plus any user header information that is also included with the message.

It is important to work out the size of the message so that the connector buffers can be set correctly, otherwise Diffusion is unable to put the messages on the wire quickly enough.

Byte pinching

With any messaging system, the smaller the messages, the lower the latency and the faster the system performs. There is a consultancy exercise that Push Technology performs as a service to analyze the messages and reduce them as much as possible. The following list includes a few of the best practices to use:

- Only send data that is required by the client.
- Look at the data format and strip any fat off the message.
- Is the information being sent a true delta?
- Client side data models

Message encoding

If you are sending large messages, it is worth compressing the messages. This happens only once on the server, and then the clients have the technology to decompress them, this also includes JavaScript clients. If other encoding is used, it is worth bearing in mind the CPU overhead required.

Client queues

A maximum queue depth can be configured for client queues so that clients are closed if their message backlog becomes too large.

The maximum queue depth must be chosen carefully as a large size might lead to excessive memory usage and vulnerability to Denial of Service attacks, whilst a small size can lead to slow clients being disconnected too frequently.

Client queues do not take any memory, as Diffusion uses a Zero Copy paradigm, but there are consequences in setting them too small or too large. If the client queue is set too small, once the client has filled its queue the Diffusion server closes the client.

When considering queue depth take into account the average message size and publication rate. Messages that are held in the client queue are not garbage collected and can get promoted, which increases their impact on GC pressure. If messages in the client queue build up, consider the maximum delay in the context of you application. For example: Assuming 100 bytes is the average message size and the application is publishing an average of 100 messages per second. If the client queue is setup to have a maximum depth of 1000 messages this means we allow messages to build up for a slow client for up to 10 seconds, during this time a slow client is building up a cache of 100k of messages to be sent.

> **Note:** It is natural for queues to build up a little with spikes in publication rate or momentary bandwidth limits, but the tolerance to such delays is expressed in the client queue depth and must be considered in that context.

Client multiplexers

Tuning multiplexers for optimal performance

The load of batching, conflating and merging messages being sent from publishers to outbound clients is spread across client multiplexers. The number of configured client multiplexers must take into account the expected message load and concurrent client connections. The more clients are assigned to a multiplexer the more load it must contend with.

By default, the number of client multiplexers is equal to the number of cores on the host system of the Diffusion server

A client multiplexer processes all client messages into the client queue. Clients are added to the multiplexers according to a round-robin load balancing policy.

Publishers either broadcast on a topic to all subscribed clients or send clients direct messages. When broadcasting all multiplexers are notified and go on to find all subscribed clients which are assigned to the particular multiplexer. When a message is sent to a particular client only that client's multiplexer is notified. It is more efficient to broadcast than it is to send the same message to a large number of clients by iterating over them.

Client multiplexers are non-blocking, high priority threads so having too many can be detrimental, as they are competing for the same resource (CPU). As a rule of thumb, the number of multiplexers must not exceed the number of available logical cores. If a client multiplexer becomes over-subscribed, message latency can increase. For maximum throughput, the

number of multiplexers can be set to the number of available cores, but this configuration is only recommended in the case where other threads are assumed to be mostly idle (for example, little inbound traffic, low publisher overhead).

Client multiplexers performance is influenced by the use of merge and conflation policies as those are executed in the multiplexer thread. It is recommended that conflation policy changes and in particular changes to merge conflation logic be profiled and written with performance in mind. In particular the use of locks or any other blocking code is highly discouraged.

Each multiplexer uses a different buffer for each output buffer size that is specified to any connector. If there were three connectors with different output buffer sizes specified, each multiplexer assigns three different buffers. Each multiplexer might also assign an extra buffer for HTTP use. A larger output buffer enables more efficient batching of messages per write, as large writes are generally more efficient but care must be taken to not overwhelm client connections regularly and causing them to be blocked for any period of time.

When a multiplexer is unable to write a message to a client because the buffer has become full, a selector thread is notified. The selector thread is responsible for watching the client and notifying the multiplexer when it becomes writable. The multiplexer remains responsible for writing the message.

Connectors

You can tune your connectors to handle multiple connections and improve performance.

Configuring multiple connectors

When there is more than one publisher application running on a server, configure a separate connector for each publisher so that buffer requirements can be specific to the connector.

It might also be beneficial to configure different connectors for different client types as their requirements can be different. This is especially true for publisher clients where there are low numbers of connections which benefit from very large buffer sizes in both directions.

Buffers

As Diffusion can have tens of thousands of connections at any one time on a machine it is important to make sure that the buffers are set correctly.

To reduce the memory footprint, the Diffusion server will condition the input and output buffers. If the buffers are set too small, Diffusion cannot write the messages in one go and delegate the task to a writer selector.

For more information, see Buffer sizing on page 704.

Thread pools

Thread pools are used within Diffusion to optimize the use of threads.

It is important to understand balance when tuning thread usage for a system. There must be sufficient thread resources required but not so many as to starve other parts of the system. At the end of the day there are only so many threads that a system can provide.

In general, when provisioning threads, separate the blocking and non-blocking activities. While it is beneficial to have more threads than cores for blocking tasks it is detrimental to the server if more threads than cores are runnable at any given time.

There are a number of places where thread pools are used within Diffusion. For more information, see Concurrency on page 652.

Configurable properties

The following key values can be configured for a thread pool to influence its behavior and use of resource:

Table 83: Values that can be configured for a thread pool

Property	Usage
Core size	The core number of threads to have running in the thread pool. Whenever a thread is required a new one is created until this number is reached, even if there are idle threads already in the pool. After reaching this number of threads then at least this number of threads is maintained within the pool.
Maximum size	The maximum number of threads that can be created in the thread pool before tasks are queued. If this is specified as 0, the pool is unbounded and so the task queue size value is ignored. Generally an unbounded pool is not recommended as it can potentially consume all machine resources.
Queue size	The pool queue size. When the maximum pool size is reached then tasks are queued. If the value is zero, the queue is unbounded. If not zero then the value must be at least 10 (it is automatically adjusted if it is not).
Keep-alive time	The time limit for which threads can remain idle before being terminated. If there are more than the core number of threads currently in the pool, after waiting this amount of time without processing a task, excess threads are terminated. A value of zero (the default) causes excess threads to terminate immediately after executing tasks.
Notification handler	A thread pool can have a notification handler associated with it to handle certain events relating to the pool. This allows for user written actions to be performed (for example, sending an email) when certain pool events (like too much task queuing) occur.

Property	Usage
	See below for more details.
Rejection handler	A thread pool can have a rejection handler associated with it to handle a runnable task that has been rejected. This allows user written actions to handle a runnable task that can not be executed by the thread pool. See below for more details.

Notification handler

A thread pool notification handler can be configured to act upon certain thread pool events.

These events are:

Table 84: Events that a thread pool notification handler can act on

Event	Description
Upper threshold reached	A specified upper threshold for the pool has been reached. This means the pool size has reached the specified size. The event is notified only once and is not notified again until the lower threshold reached event has occurred.
Lower threshold reached	A specified lower threshold for the pool has been reached after an upper threshold reached event has been notified. This means the pool size has now shrunk the specified size.
Task rejected	The pool has rejected a task because there are no idle threads available and the task queue has filled. What happens to the rejected task depends upon the type of pool. Typically, the task is run within the thread that passes the task to the pool, which is not desirable. This is why the thread ought to be notified when it occurs. This differs from the rejection handler in that it does not expose the runnable task. This means it can be used only for notification.

The notification handler is a user written class which must implement the `ThreadPoolNotificationHandler` interface in the threads Java API. The name of such a class can be configured for in-bound or out-bound thread pools or for connector thread pools in which case an instance of the class is created (and must have a no arguments constructor) when the thread pool is created.

Rejection handler

A thread pool can have a rejection handler associated with it to handle a runnable task that has been rejected.

Two rejection handlers are provided with Diffusion. These are the `ThreadService.CallerRunsRejectionPolicy` and `ThreadService.AbortRejectionPolicy`.

The `ThreadService.CallerRunsRejectionPolicy` executes the runnable task in the thread that tried to pass the runnable task to the thread service. The `ThreadService.CallerRunsRejectionPolicy` does not execute the task and instead generates an exception.

Note: The inbound thread pool has a mandatory caller runs rejection policy.

The rejection handler is a user written class which must implement the `ThreadPoolRejectionHandler` interface in the threads Java API. The name of such a class can be configured for in-bound or out-bound thread pools or for connector thread pools in which case an instance of the class is created (and must have a no arguments constructor) when the thread pool is created.

Adjusting the configuration

Adjust thread pools gradually. Ideally, duplicate expected maximum loads in test environment. This environment can be used to tune the thread pools to satisfy the load. Tune the thread pools so they are just able to cope with the maximum load, increasing them beyond this might degrade overall performance.

Background thread pool:

In general, the defaults suffice for the tasks assigned to the background thread pool by Diffusion. If you assign tasks to the pool yourself, consider increasing the number of threads.

Inbound thread pool:

This pool is used to handle inbound connections and messages. Increasing the thread pool allows new connections and received messages to be handled over a greater number of threads. However, much of the behavior in this pool can involve locking the clients or parts of the topic tree. This can cause lock contention that delays processing.

Due to the underlying implementation of Java NIO sockets a high rate of threads being added/removed from the incoming thread pool will result in the allocation of off-heap byte buffers. In extreme cases this can result in an out of memory exception being thrown as the server runs out of off heap allocation space.

Client reconnection

You can configure the client reconnection feature by configuring the connectors at the Diffusion server to keep alive the client session.

Normally when a client application loses its connection to the server, perhaps due to some communications error, the only option is for it to connect again and then re-establish the state of the topics to which it was subscribed. There is however, the facility for clients to be able to reconnect a lost connection without losing its topic state or messages that were queued for it whilst it was not connected.

This can be useful for mobile clients where connections are less reliable.

Server configuration

To enable clients to reconnect, connectors must be configured to keep alive client sessions that are in the DISCONNECTED state. To do this a keep-alive time must be specified for the connector.

Specify how long a disconnected client's session is kept alive on the server and how many messages to queue for a client while it is disconnected by using the `<reconnect>` element in the `etc/Connectors.xml` configuration file.

```
<connector>
 ...
 <reconnect>
  <keep-alive>60s</keep-alive>
  <max-depth>1000</max-depth>
 </reconnect>
 ...
</connector>
```

A client can reconnect to the server through this connector within 60 seconds of becoming disconnected. While the client is disconnected, up to 1000 messages are queued for it. These messages are delivered to the client when it reconnects.

If a client signals that it wants to disconnect, the client state on the server is removed when the client disconnects. However, in all other circumstances where the client loses connection, the client goes into the DISCONNECTED state, where the subscriptions are retained and messages are queued as normal for the amount of time specified by the keep-alive time of the connector.

If the server is configured to expect reconnecting clients, clients that are not currently connected and might reconnect are excluded from the regular system pings that the server sends to clients. The reason for this behavior is that the keep-alive time might be longer than the interval between system pings. If this is the case, the reconnecting client is unable to respond to the pings from the server. Hence, no pings are sent from the server to a reconnecting client.

If the client then reconnects before the keep alive period has expired, the sending of messages to the client resumes from the point when the failure occurred. Messages that were in transit at the time of disconnection might be lost.

The only way to ensure the delivery of messages on reconnection is for the publisher to mark the messages as requiring acknowledgment as any such messages that have been sent but not acknowledged on reconnection are requeued for the client. The delivery of acknowledged messages is assured. However, because an ack from the client might have been lost during the disconnection, there is the possibility that a message might be delivered to the client twice in this scenario.

It is important when using acknowledged messages to ensure delivery after reconnect that the ack timeout set for messages is sufficiently long to allow for the time that a message can be queued for a client plus the keep alive time configured for reconnection.

If an ack timeout expires before a message is even dequeued for a client, the non-acknowledgment is notified and the message is not sent to the client.

Message queue management

Managing message queues when using client reconnection

When a client session is being kept alive, messages for the client continue queuing for the client until the keep alive period expires or the client reconnects. This puts an unusual load on the client queue and the facility exists to adjust the maximum client queue depth for the period of disconnection.

This is done by specifying a keep alive queue depth which is greater than the normal maximum queue depth. When disconnected the queue can expand to the higher value and when reconnection occurs and the queue starts to drain, when the queue size goes down to a value of 80% of its previous limit, the maximum queue depth reverts to the normal value.

The keep alive queue depth has an effect only if it represents a value higher than the normal maximum queue depth.

Client reconnection

Configuring the reconnection of clients

Not all clients support reconnection but those that do have a reconnect method which they can call on notification of a lost connection.

If the reconnection succeeds, the client is subscribed to all of the same topics as before and starts to receive messages again, including all of those queued whilst the client was disconnected.

Messages in transit at the time of disconnection might be lost, however any message marked as requiring acknowledgment and sent by the server that was not acknowledged by the client is retransmitted on reconnect. The delivery of acknowledged messages from client to server is assured on reconnect although there is the possibility that the client might receive a message it had acknowledged before the connection again after reconnection if the ack had never reached the server.

A reconnection might not succeed, either because keep alive is not specified on the connector that the client has connected to, or the keep alive time period has expired. In this case a normal new connection is established with the same topic set as was specified on the original connection.

How to test reconnection in my environment?

To simulate a communication error, we use a proxy between the client and the server.

- Start Diffusion. (By default it uses port 8080)
- Set the proxy to listen on a different port (for example, 9090) and redirect the connection to 8080
- Connect the client through the proxy on port 9090.
- Kill the proxy.
- Start the proxy.
- Reconnect the client.

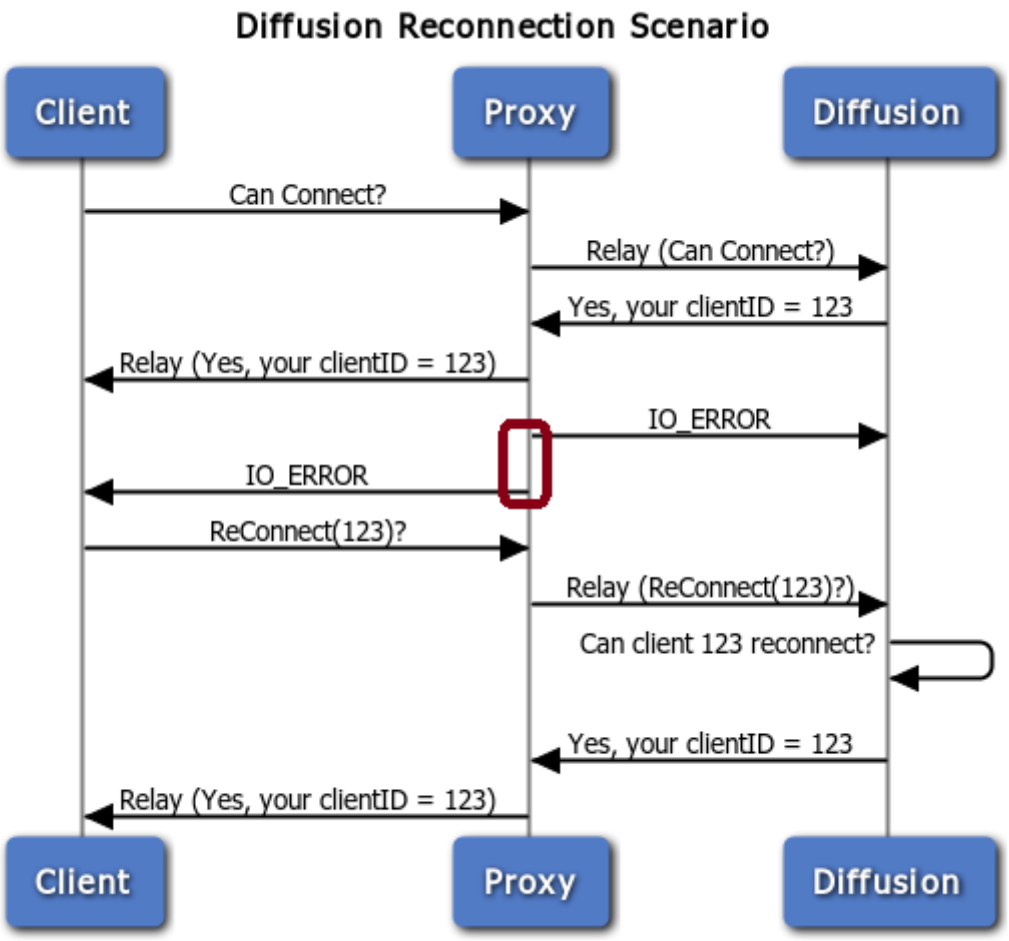


Figure 84: Reconnection scenario

Note: The proxy behaviors are different depending on the operating system and the TCP/IP stack configuration.

Common errors

1. From the client, request a connection close and call reconnect.

   After a close request, the client cannot reconnect. In this case, the client will establish a new connection with a different client ID assigned by Diffusion.
2. Unplug the network wire from the computer where the client is running

   This will not throw an IO_ERROR in the other end of the connection.

Related Links

Reconnecting with the ActionScript API on page 517
The ActionScript API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

Reconnecting with the JavaScript API on page 510

The JavaScript API supports reconnection. If you have reconnection enabled and you lose your connection to a server, you can reestablish it, using the same client ID and with the client subscriptions preserved.

Client failover

You can configure a client to failover to another Diffusion server after it loses connection to the Diffusion server it was previously connected to.

Client failover is when a client loses its connection to a server and attempts to connect to a different one. The client is provided with a list of servers. If a client loses its connection to a server it can automatically attempt to connect to the next server in the list. If it fails to connect or loses its connection to that server, it tries the next server on the list. This is referred to as autofailover. Generally the list of servers to connect to must be provided before attempting to make the connection. How the list of servers is provided differs between client APIs and the JavaScript client does not support autofailover but it can be implemented using the callback methods.

Using automatic failover

If a client has an established connection that it loses, autofailover attempts to open a new connection in the next connection in the list. This is not compatible with reconnection because reconnection attempts to preserve the state of the client (the client ID and the subscribed topics). As the new server has no knowledge of the client it is unable to preserve this state. Autofailover must be enabled and a list of servers to connect to provided.

Using load balancing with autofailover

You can enable load balancing in conjunction with autofailover. When load balancing is enabled and a client loses connection, the list of servers is shuffled before the client selects the next server to attempt to connect to.

In Java, for example, you can enable load balancing by using the `setLoadBalancing` method on the `ConnectionDetails` object.

Using server cascading

When a client attempts to place a connection, if the attempt fails, the next server in the list is chosen. Server cascading is similar to autofailover except this logic is applied prior to a connection, whereas autofailover applies once a connection is in place.

In Java, for example, you can enable server cascading by using the `setCascading` method on the `ConnectionDetails` object.

> **Note:** Server cascading is different to protocol cascading, which attempts to connect to the same server using different protocols before a connection has been opened.

Java failover

Configuring failover in Java

In Java the ConnectionDetails object supports a collection of ServerDetails objects. The server details are use the control failover between servers. The ConnectionDetails factory methods provide several options for creating ConnectionDetails with multiple ServerDetails objects. A collection of ServerDetails object can be passed as a parameter, a varargs method supports ServerDetails and another varargs method supports

String URLs, which ServerDetails objects are constructed from. After construction the ServerDetails objects used by the ConnectionDetails can be altered by calling the `setServerDetails(Collection<ServerDetails>)` method.

The following code supports autofailover from the server with the IP address 192.168.0.1 to the server 192.168.0.2. If the client loses connection to 192.168.0.1 it tries to connect to 192.168.0.2. It uses the varargs method to create a ConnectionDetails object with multiple ServerDetails objects constructed from String URLs.

```
ConnectionDetails details =
    ConnectionFactory.createConnectionDetails(
        "dpt://192.168.0.1:8080",
        "dpt://192.168.0.2:8080");
details.setAutoFailover(true);
ExternalClientConnection client = new
 ExternalClientConnection(listener, details);
client.connect();
```

For further information refer to the Java API documentation for ConnectionDetails and ServerDetails.

JavaScript failover

Using failover in JavaScript

The JavaScript client does not support autofailover. Support for failover is limited. If the connection attempt fails DiffusionClient.connect(DiffusionClientConnectionDetails) can be called with a different object. You must provide the logic to do this on connection failure.

ActionScript failover

Using failover in ActionScript

In ActionScript the ConnectionDetails object supports an array of ServerDetails objects. The server details are use the control failover between servers. The ConnectionDetails constructor has a mandatory ServerDetails object. After construction additional ServerDetails objects used by the ConnectionDetails can be altered by calling the `addServerDetails(ServerDetails)` method and the `setServerDetailsArray(Array)` method.

This code:

```
var server0:ServerDetails = new
 ServerDetails("dpt://192.168.0.1:8080");
var server1:ServerDetails = new
 ServerDetails("dpt://192.168.0.2:8080");
var details:ConnectionDetails = new ConnectionDetails(server0);
details.addServerDetails(server1);
details.setAutoFailover(true);
var client:DiffusionClient = new DiffusionClient();
client.connect(details);
```

supports autofailover from the server with the IP address 192.168.0.1 to the server 192.168.0.2. If the client loses connection to 192.168.0.1, it tries to connect to 192.168.0.2. It uses the `addServerDetails(ServerDetails)` method to add a single additional server to connect to. For further information refer to the Flex Classic API documentation for ConnectionDetails and ServerDetails.

Client throttling

Throttling is a method of ensuring that the Diffusion server limits (throttle) the volume of messages it transmits to a client within a specified period of time. This can be used to limit bandwidth usage or to prevent more messages being sent to a client than it can cope with.

How does throttling work?

Throttling is applied to a client queue so that the number or volume of messages sent to that client is restricted. Diffusion only dequeues a message to send to a client if that client has not breached its throttling limit.

Throttle types;

- Messages per second (Only a specified number of messages are sent every second.)
- Bytes per second (Only a specified number of bytes are sent every second)
- Message interval (A single message is sent every n milliseconds.)
- Buffer interval (A full output buffer (or the equivalent of) is sent every n milliseconds)

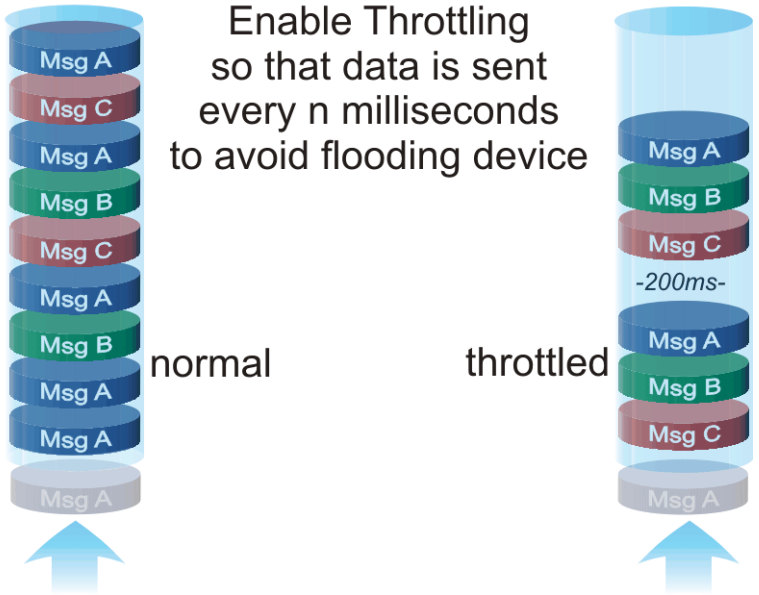


Figure 85: Normal and throttled client queues

Enabling throttling

Throttling can be enabled on a client-by-client basis from within the publisher.

To throttle a `Client.throttle` method which allows you to specify the type of throttling and a limit. A `clientThrottler` reference is returned.

If the throttle method is called for a client that is already throttled, it has the effect of removing the old throttler and adding a new one.

Call the `Client.removeThrottler` method to stop throttling.

The `Client.isThrottling` method can be used to determine whether a client is currently throttled and, if it is, the getThrottler method can be used to determine the type of throttling and the current limit.

Memory considerations

Memory usage in the Java VM

Garbage collection (Java HotSpot VM)

Typically you do not have to alter the default garbage collection for the Java VM, but, depending upon the nature of the application, better performance and memory usage might be obtained by tuning the VM's garbage collection parameters.s

Garbage collection tuning is a complex area and consult the Oracle guides to garbage collection tuning. However, some considerations are mentioned below:

Generational vs incremental collection

The default garbage collection algorithm used is generational which means it views newer object references differently from older ones and assumes that what is to be collected is generally in the newer space.

This approach is best for most cases, but in the case where Diffusion is maintaining message queues for clients, messages might remain in a queue for long enough to be moved out of the space eligible for regular collection. The overall effect of this might be that memory (heap) usage appears to grow indefinitely and a sudden peak of memory usage might result in an out of memory error. This can be overcome by use of the incremental garbage collector which will do period sweeps of older objects. You can do this by using the -Xincgc flag for the Diffusion Java VM.

Platform-specific issues

To run Diffusion it might be necessary to increase the number of sockets and reduce timewait.

It might also be necessary to increase the number of open files that is allowed on UNIX or Linux systems

Socket issues

To fix these problems, complete the following steps based on platform.

Windows

Setting values on Windows

Setting TCP timed wait

This parameter determines the length of time that a connection stays in the TIME_WAIT state when it is closed. When a connection is in the TIME_WAIT state, the socket pair cannot be reused. This is also known as the 2MSL state because the value is twice the maximum segment lifetime on the network. See RFC 793 for further details.

Add TcpTimedWaitDelay registry values as a workaround. You can set these values through REGEDIT command.

Set TcpTimedWaitDelay to 30:

1. Select **Start** > **Run**.
2. In the available field, enter `regedit`.

3. Go to the key directory file: `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters/TcpTimedWaitDelay`. The value type is `REG_DWORD`.
4. Double-click **TcpTimedWaitDelay**.
5. Select **Decimal**.
6. Type `30` in the **Value** data field. The default value for this field is 0xF0 (240 decimal). The valid range is 30-300 (decimal).

Setting MaxUserPort

This parameter controls the maximum port number used when an application requests any available user port from the system. Normally, short-lived ports are allocated in the range from 1024 through 5000. Setting this parameter to a value outside of the valid range causes the nearest valid value to be used (5000 or 65534).

Add MaxUserPort registry values as a workaround. You can set these values through REGEDIT command.

Set the MaxUserPort to 65534

1. Select **Start** > **Run**.
2. In the available field, enter `regedit`.
3. Go to the key directory file: `HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters/MaxUserPort`. The value type is `REG_DWORD`.
4. Double-click **MaxUserPort**.
5. Select **Decimal**.
6. Type `65534` in the **Value** data field. The default value for this field is 0x1388 (5000 decimal). The valid range is 5000 – 65534(decimal).

Linux

Configuring sockets values on Linux

Decrease the time wait before closing the sockets by entering:

```
# echo 3 > /proc/sys/net/ipv4/tcp_fin_timeout
```

Sometimes systems are now configured to prevent one from using a large number of ports, check the port range and modify if required.

```
# cat /proc/sys/net/ipv4/ip_local_port_range
```

This can be increased by issuing the following command

```
# echo "1025 65535" > /proc/sys/net/ipv4/ip_local_port_range
```

To have these new values take effect you might have to do (as root)

```
# /etc/rc.d/init.d/network restart
```

If you want these new values to survive across reboots you can at them to `/etc/sysctl.conf`.

```
# Allowed local port range
net.ipv4.ip_local_port_range = 1025 65535
# net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_fin_timeout = 3
```

UNIX

Increasing the number of files a process on a UNIX system can open also increases the number of sockets that process can open. The operating system uses file descriptors to handle filesystem files as well as pseudo files, such as connections and sockets.

You need the following number of sockets for each client connection:

- DPT, WS - one socket per connection
- HTTP Full Duplex, HTTP Polling, HTTP Chunked Streaming - two sockets per connection

Use the `ulimit` command to increase the number of open files. You can do this in one of the following ways:

- As a global setting.

  This can be set by your network administrator.
- In the start script for Diffusion.

  Edit the *`diffusion_installation`*`/bin/diffusion.sh` file to add the following line at the start:

  ```
  ulimit -n open_files
  ```

  Where the value of *open_files* is any suitable integer value, for example 8192.

You can use the `java.lang:type=OperatingSystem` MBean to inspect the number of files on your UNIX operating system. See the following properties:

`MaxFileDescriptorCount`
: The total number of files that a process on a UNIX system can open. This is the number that you can set with `ulimit -n`.

`OpenFileDescriptorCount`
: The number of files that are currently open.

The difference between these values is the number of files you have available to use for sockets.

Publisher design

Considerations when designing a publisher

Consider the following points when designing and writing a publisher:

Data modeling
: The way that the data is fed to a publisher and the way in which the state of the data is maintained within a publisher is key to good performance. Keep message sizes to a minimum and this can be achieved using fine data granularity enabled by hierarchic topics.

Caching
: Cache messages wherever possible rather than building new ones every time one must be sent. This particularly applies to topic load messages which can be cached to send to every new client that subscribes, and rebuilt only when the data actually changes. The ideal place to keep such cached messages is with a data object attached to the topic (see topic data pattern).

String handling

Building of Strings by concatenation is very inefficient in Java. Keep String concatenation to a minimum. When String content is used, message caching can help to some degree and wherever possible cache Strings that must be built.

Conditional processing

Excessive use of conditional processing (Checking of topic names, and so on) can be expensive. Use of the topic data pattern can significantly reduce the need for such processing when many topics are in use.

Topic naming

As every message must carry the topic name, long topic names can lead to a large amount of data traffic which can be disproportional to the data being carried. This can be a particular problem when using hierarchical topics. A solution is to use the topic aliasing feature so that only short aliases of the topic names are transmitted in messages.

Concurrency

Concurrent programming means that access to data often must be synchronized but care must be taken not to synchronize more than is necessary as performance can be significantly affected.

Diagnostics

Diffusion provides various mechanisms for logging information and diagnosing problems. The major internet browsers also include useful diagnostic tools. All are discussed in this section.

Logging

The Diffusion server logs messages about its activities and any problems that occur. Log messages are written to the console (the process output, which can appear in a terminal window) and to a log file. The level of logging can be adjusted, from a moderate rate of messages under normal circumstances for system monitoring, to very detailed logging for problem diagnosis.

Configuring logging

The log messages created by the Diffusion server, and by publishers deployed to the server, are filtered by the configuration in `etc/Logs.xml`.

You can configure the following aspects of logging:

- The level of logging to the console
- The level of logging to a file
- The name and location of the file
- Whether the log files rotate based on time or file size or both
- The time interval to use to rotate the files
- The file size to use to rotate the files
- The number of old log files to keep

Application code that uses the Diffusion API but is not deployed in a server environment (that is, code that is not part of a publisher) can still use the logging facilities. In this case, the API can be used to reconfigure the logging back-end, and `etc/Logs.xml` configuration does not play a part.

Log levels

The log levels, in severity order, are as follows:

Table 85: Log levels

Configured value	Messages that are written to logs (SLF4J values)
ERROR	ERROR messages only.
WARN	WARN and ERROR messages.
INFO	INFO, WARN, and ERROR messages.
DEBUG	DEBUG, INFO, WARN, and ERROR messages.
TRACE	TRACE, DEBUG, INFO, WARN, and ERROR messages.

Warning: Logging can use considerable CPU resources Enable only significant log messages (INFO and above) in a production environment. Performance degrades significantly when running at finer logging levels as more messages are produced, each requiring processing.

Log format

From Diffusion 4.6, the log format is formally documented. This eases integration with log monitoring tools.

Each log line is made up of a number of fields. All of the fields except for the Exception are formatted on a single line, delimited by pipe (|) characters.

```
2013-07-17 15:21:57.355|INFO|main|PUSH-000165|Diffusion Server
 started.|com.pushtechnology.diffusion.DiffusionController
```

If a log event has an exception, the exception message and stack trace directly follows the message line.

```
2013-07-17 16:57:50.659|WARN|main|PUSH-0000258|
Failed to load publisher 'ConsolePublisher'.|
com.pushtechnology.diffusion.publisher.PublisherManagerInstance
com.pushtechnology.diffusion.DiffusionException: Class
 com.pushtechnology.diffusion.apps.console.publisher.ConsolePublisher
 for Publisher ConsolePublisher not found
 at ...PublisherManagerInstance.createInternalPublisher(Publisher...)
 at ...PublisherManagerInstance.loadPublisher(PublisherManagerInstance.java:
 at ...PublisherManagerInstance.loadPublishers(PublisherManagerInstance.java
 at ...PublisherManager.loadPublishers(PublisherManager.java:51)
 at ...DiffusionController.startImpl(DiffusionController.java:220)
 at ...DiffusionController.start(DiffusionController.java:126)
 at ...api.server.DiffusionServer.start(DiffusionServer.java:105)
 at ...api.server.DiffusionServer.main(DiffusionServer.java:140)
 at ...Diffusion.main(Diffusion.java:20)
```

The meaning of each field is described in the following table.

Table 86: Fields included in the logs

Field	Optional or Mandatory	Format/values stable between releases	Description
Time stamp	Mandatory	Yes	The time and date the log event occurred. Asynchronous logging is enabled by default. The server might log a message in a different thread to the one that produced the log event, and at a slightly later time. Consequently, log lines might not be logged in exact time stamp order. The time stamp is displayed using the timezone configured for the JVM running the server. The date format can be changed in the `Server.xml` configuration file.
Severity	Mandatory	Yes	The log severity, using the SLF4J levels: `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`.
Thread	Mandatory	No	The name of the Java thread that logged the event.
Code	Optional	Yes	A code identifying the message type. All documented messages have a code. Example: `PUSH-000123`. These messages can be logged at any level, but are typically `INFO` or above.
Message	Mandatory	No	A natural language description of the event.

Field	Optional or Mandatory	Format/values stable between releases	Description
Logger name	Mandatory	No	The logger name. Usually the fully qualified name of the Java class that produced the event.
Exception	Optional	No	If the log event has an associated Java `Throwable`, the exception message and stack trace directly follows the message line.

Optional fields are empty if the log event does not have the information.

The third column indicates whether fields are stable between releases. Where possible, Push Technology will not change the format or values of these fields so they can be relied on for automated log monitoring. The fields not marked as stable are more likely to change between releases, including patch releases.

Log headers

Currently, Diffusion log files start with a special header.

```
2015-02-04 11:30:33.841 : Starting log for Diffusion 5.2.0_dev
 (Server) (2015-02-04 11:19:31)
```

Do not depend on this header. In a future release, this header will be replaced with a standard log message.

Log stopped

When the logging rotates, it outputs the message `Log stopped` at the end of the log file before creating a new log file and continuing to log messages in that new file.

Related Links

Basic integration with Splunk on page 699
How to achieve basic integration between Diffusion and the Splunk™ analysis and monitoring application

Log Messages on page 735

Logging API

The Diffusion Java API includes a logging facility that can be used by applications to log information.

Prefer SLF4J

From Diffusion version 4.6, the product implementation has switched to the API logging SLF4J.

In this release, the `LogWriter` API described below is still supported for application use. We recommend that new applications use SLF4J to log events instead of `LogWriter`.

Whether SLF4J or `LogWriter` is used, log events are processed by a shared logging back-end and are written to the same destinations.

For more information, see SLF4J.

DEPRECATED: LogWriter API

Diffusion provides its own `LogWriter` interface that provides similar functionality to `java.util.logging.Logger` in the JDK. We recommend that new applications use SLF4J to log events instead of `LogWriter`.

Mapping of LogWriter log levels

`LogWriter` has a method for each level of logging. The method you use determines the log level at which the message is logged. For example, if you use the `fine` method, the message is logged only if `FINE` logging (or lower) is enabled in the configuration.

These log levels are mapped to the SLF4J log levels, which are used in the log files. The following table describes the mapping of the `LogWriter` methods to the level used in the log files.

Table 87: Mapping between LogWriter methods and Diffusion log levels

`LogWriter` method	Level used in log file
`severe()`	`ERROR`
`warning()`	`WARN`
`advice()`	`INFO`
`info()`	`INFO`
`fine()`	`DEBUG`
`finest()`	`TRACE`

The logging back-end

SLF4J and `LogWriter` are both APIs that separate the code used to log a message from the logging implementation. The work of formatting and writing out messages is done by the logging back-end.

In this release, the Diffusion logging back-end is based on `java.util.logging`. The back-end implementation of logging might change in a future major release. The information in this section is subject to change.

In a Diffusion server environment there are two log handlers – implementations of `java.util.logging.Handler`. One handler is responsible for writing messages to the console and the other for writing messages a file. In a non-server environment, there is a single default handler that logs to `System.err`.

In either case, additional handlers can be added to process log messages in any way that is required. For example, you might configure a logger that sends messages directly to a UNIX `syslog` process.

A handler can be added using the `LogWriter.setHandler()` method. You can add a handler that processes messages from a single publisher by adding it to the `LogWriter` returned by its `getLogger()` method. Alternatively you can add a handler to the root `LogWriter` obtained from `Logs.getLogger()` and it handles messages for all loggers, including those used by the Diffusion software.

In a server environment, additional handlers do not replace the default handlers. In a non-server environment, the first handler added to the root `LogWriter` replaces the default handler.

Logging in the .NET API

The .NET API produces logging information. The logging facility uses NLog.

These logging instructions apply to both the Unified API and the Classic API.

NLog is included in the .NET client assembly. For more information about logging with NLog, see https://github.com/NLog/NLog/wiki/.

Logging basics

The NLog `Logger` class acts as the source of the log messages. In general, use one logger per class and pass in the name of the class as the logger name.

To log a message at a certain level use the write methods provided by the `Logger` class.

The following log levels are provided:

- Fatal
- Error
- Warn
- Info
- Debug
- Trace

These levels are listed in order from most severe to least severe.

Configuring the log output

You must configure NLog to output the log messages produced by your application to a target or targets. NLog provides a large number of targets for your output, including File and Console. NLog also enables you to create your own targets.

In addition to specifying targets for the log output, use NLog configuration to define rules that specify to which target log messages with particular levels or logger names are directed.

You can configure NLog in the following ways:

- Using a configuration file, `NLog.config`, that is located in the same directory as your client application.

  For more information, see https://github.com/NLog/NLog/wiki/Configuration-file.
- Using the Configuration API to configure NLog in your application code.

  For more information, see https://github.com/NLog/NLog/wiki/Configuration-API.

Connection counts

The Diffusion server produces connection summaries.

At one minute past midnight Diffusion creates an entry in the file `logs/ConnectionCount`, and resets the counter.

The value in the third column is the total number of client connections that have been made that day.

The value in the fourth column is the maximum number of concurrent connections that have been active that day.

The scheduling of this reset defaults to once per day, but it can be controlled by supplying the `<reset-frequency>` element in `etc/Statistics.xml`.

An example is shown here:

```
2013-03-27 00:01:40     4.4.0_05     128     31
2013-03-28 00:01:48     4.4.0_05     139     28
2013-03-29 00:01:56     4.4.0_05     177     28
2013-03-30 00:01:05     4.4.0_05     118     41
2013-03-31 00:01:13     4.4.0_05     196     30
2013-04-01 00:01:22     4.4.0_05     207     36
2013-04-02 00:01:31     4.4.0_05     188     19
2013-04-03 00:01:41     4.4.0_05     244     44
2013-04-04 00:01:41     4.4.0_05     188     26
2013-04-05 00:01:41     4.4.0_05     195     39
```

JavaScript diagnostics

This section describes how to enable and make use of the browser consoles.

There are currently more than 30 web browsers in use. This section discusses some of the more widely used browsers, namely Firefox, Chrome, Internet Explorer, Opera, and Safari.

As JavaScript is a typeless language, it can very difficult to debug. DiffusionClient surrounds both onMessage and topic listener function invocations with a try..catch block. This might mean that if there is an error in the functions that accept WebClientMessage, this might silently fail and not produce the expected results. To trap this, there is a DiffusionClient.trace(message) function. The trace function creates a message which can be seen in the browser's logging console. Runtime errors are also displayed within the console.

To allow debugging of the client, include the following entry:

```
<script type="text/javascript"> function startDiffusion()
 { DiffusionClient.setDebugging(true); } </script>
```

Mozilla Firefox

Firebug integrates with Firefox allowing you to debug and monitor JavaScript, and edit HTML and CSS. Firebug requires Firefox 3.5 or higher and can be downloaded from the following location: https://addons.mozilla.org/en-US/firefox/addon/1843

To start debugging, click on the Firebug icon in the bottom right corner of the screen.

Ensure that logging is enabled in the drop-down menus for **Console** and **Net**

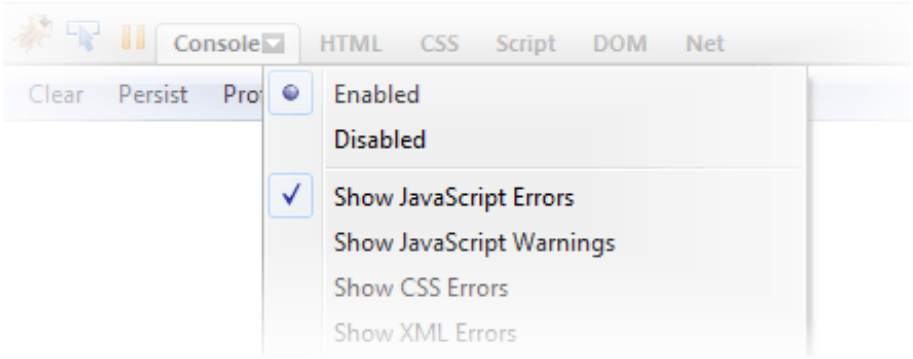


Figure 86: Firefox Console

Google Chrome

Chrome's console can be found in the **Developer** > **Developer tools** section of the menu.

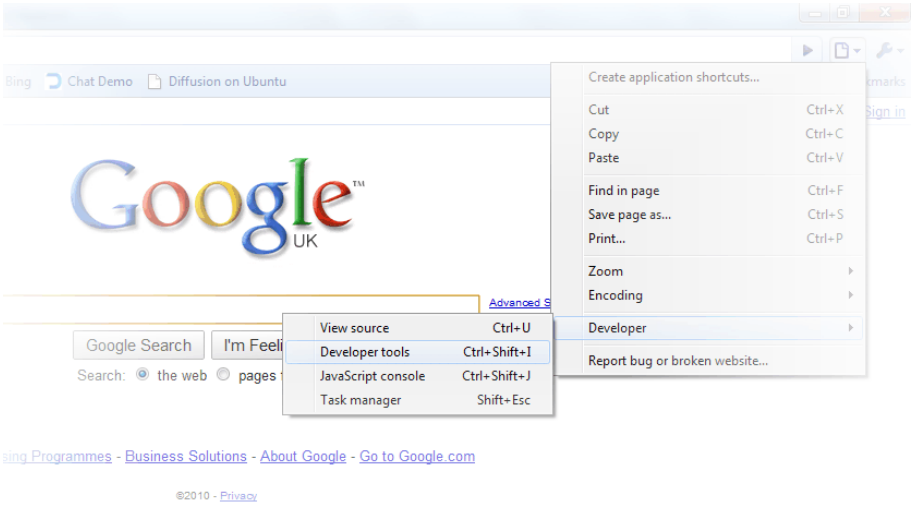
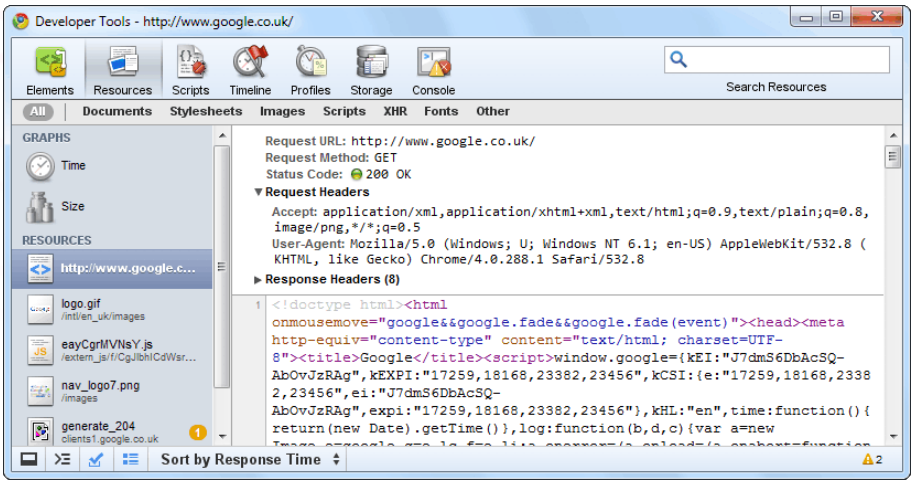


Figure 87: Chrome's console

On the console, go to the **Resource** tab and click **Enable**.



Internet Explorer

IE8 has a console which is accessed by selecting **Tools** > **Developer Tools** of by pressing F12.

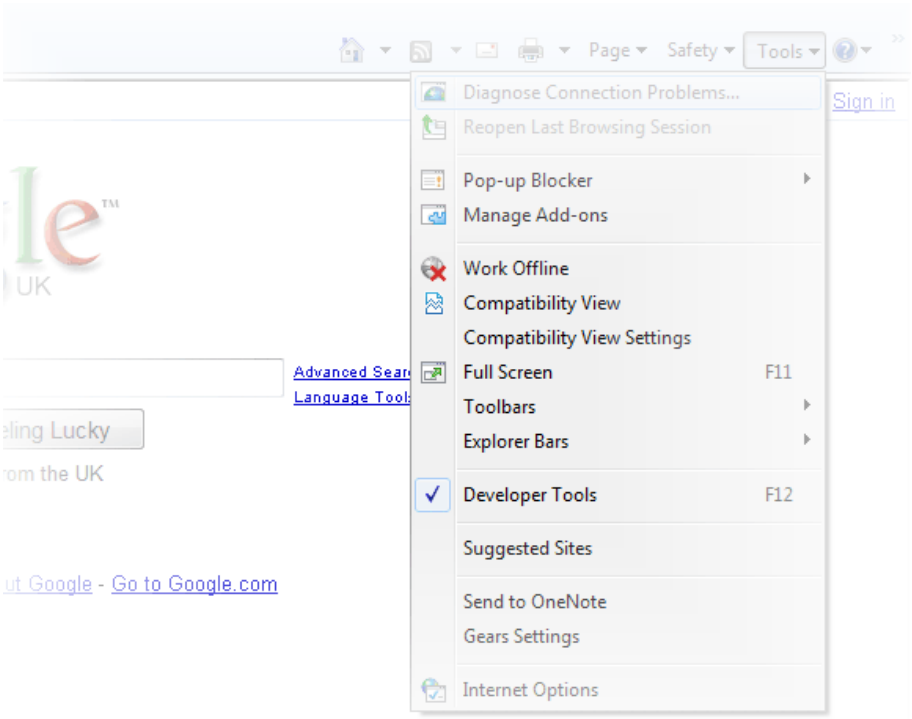


Figure 88: Internet Explorer console

Doing this brings up the console. In some cases when attempting to start Developer tools, the window fails to appear when it is outside the viewport. This can be remedied by pressing `Alt` +Space and choosing **restore**.

Opera

The Opera console is similar to that of Firefox and is accessed through the **Tools** > **Advanced** > **Developer Tools** menus.

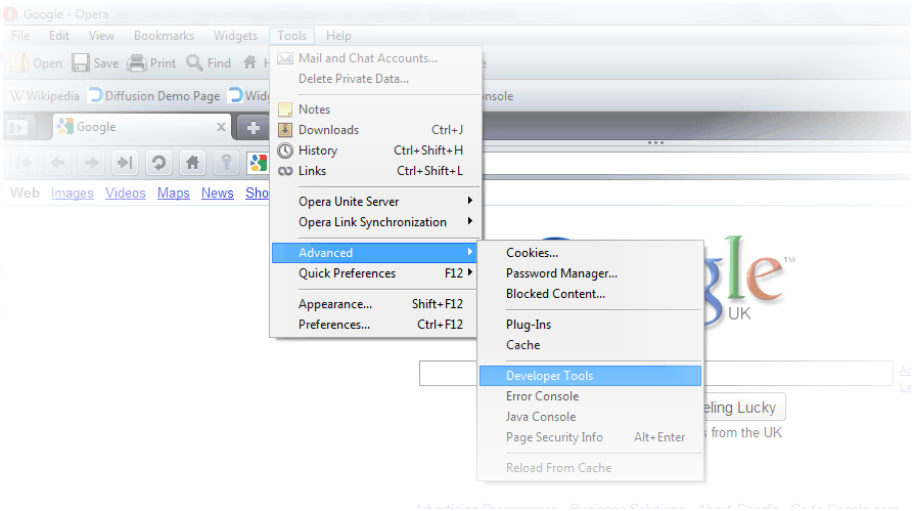
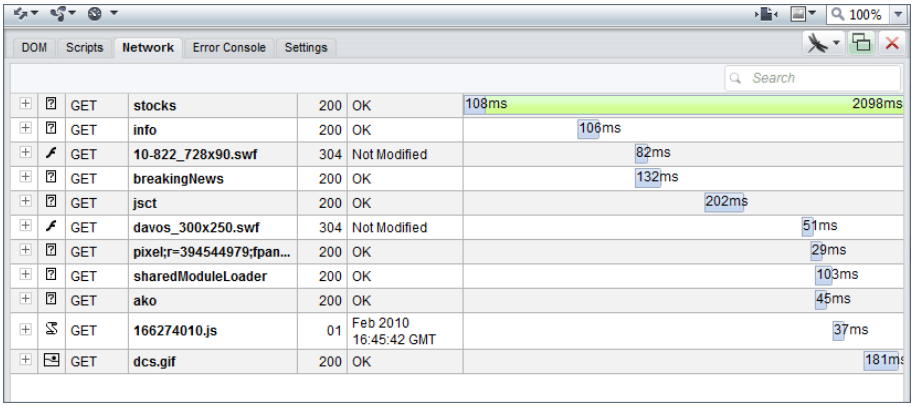


Figure 89: Opera console

Doing this opens the console that enables you to begin debugging.


100%
DOM Scripts Network Error Console Settings
Search
GET stocks 200 OK 108ms 2098ms
GET info 200 OK 106ms
GET 10-822_728x90.swf 304 Not Modified 82ms
GET breakingNews 200 OK 132ms
GET jsct 200 OK 202ms
GET davos_300x250.swf 304 Not Modified 51ms
GET pixel;r=394544979;fpan... 200 OK 29ms
GET sharedModuleLoader 200 OK 103ms
GET ako 200 OK 45ms
GET 166274010.js 01 Feb 2010 16:45:42 GMT 37ms
GET dcs.gif 200 OK 181ms

Safari

Safari's console is Web Inspector, and functions in a very similar way to Chrome's Developer Tools. Profiling and Resources. Enabled it at first use

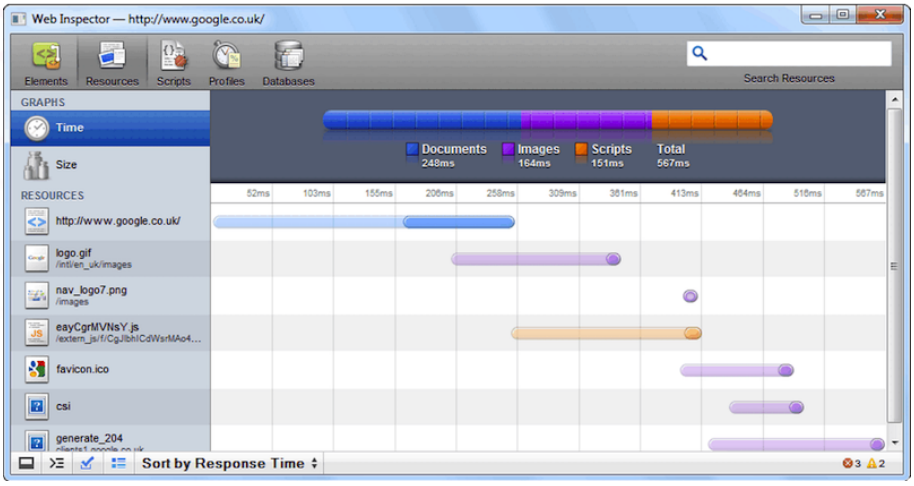


Figure 90: Safari's console

The DiffusionClient library has been compressed. If you want to use the DiffusionClient uncompressed version, use

```
<script type="text/javascript" href="lib/DIFFUSION/
diffusion.uncompressed.js"></script>
```

Flex and Flash diagnostics

For the Flash connectivity, it is important to got the policy files correct. If the policy file is not loaded or has the incorrect syntax or values, connectivity silently fails. You can get more information by following the steps below.

Diffusion also provides a debug-friendly version of the Flash client libraries: `diffusion-flex-debug-`*`version`*`.swc`, where *version* is the Diffusion version number, for example 5.5.3. You can use this to output additional debug information, such as line numbers, in any stack traces that you experience.

Flex/ActionScript diagnostics

1. Download the debug version for the Flash Player. Flash player debug versions are available at the following location: http://www.adobe.com/support/flashplayer/downloads.html
2. Create a `mm.cfg` file

The table below shows where to create a `mm.cfg` file

The following text is an example of entries in a `mm.cfg` file.

```
# Enables policy file logging
PolicyFileLog=1
# Optional; do not clear log at startup
PolicyFileLogAppend=0
ErrorReportingEnable=1
TraceOutputFileEnable=1
```

Viewing the log files

In the following directory there can be the following files: `flashlog.txt` and `policyfiles.txt`

Table 88: Location of the `flashlog.txt` file

Windows 95/98/ME	`%HOMEDRIVE%\%HOMEPATH%`
Windows 2000/XP	`C:\Documents and Settings \username`
Windows Vista	`C:\Users\username`
OS X	`/Library/Application Support/ Macromedia`
Linux	`/home/username`

Table 89: Location of the `policyfiles.txt` file

Windows 95/98/ME/2000/XP	`C:\Documents and Settings \username\Application Data \Macromedia\Flash Player\Logs`
Windows Vista	`C:\Users\username\AppData \Roaming\Macromedia\Flash Player \Logs`
OS X	`/Users/username/Library/ Preferences/Macromedia/Flash Player/Logs`
Linux	`home/username/Macromedia/ Flash_Player/Logs`

Debugging a publisher

Creating a project in Eclipse for debugging publishers.

After installing the Diffusion server, create a Java source directory (`src`) inside the Diffusion installation directory. Inside the `src` directory, create a `java` directory.

Create a new Eclipse Java Project and make the project root the src directory.

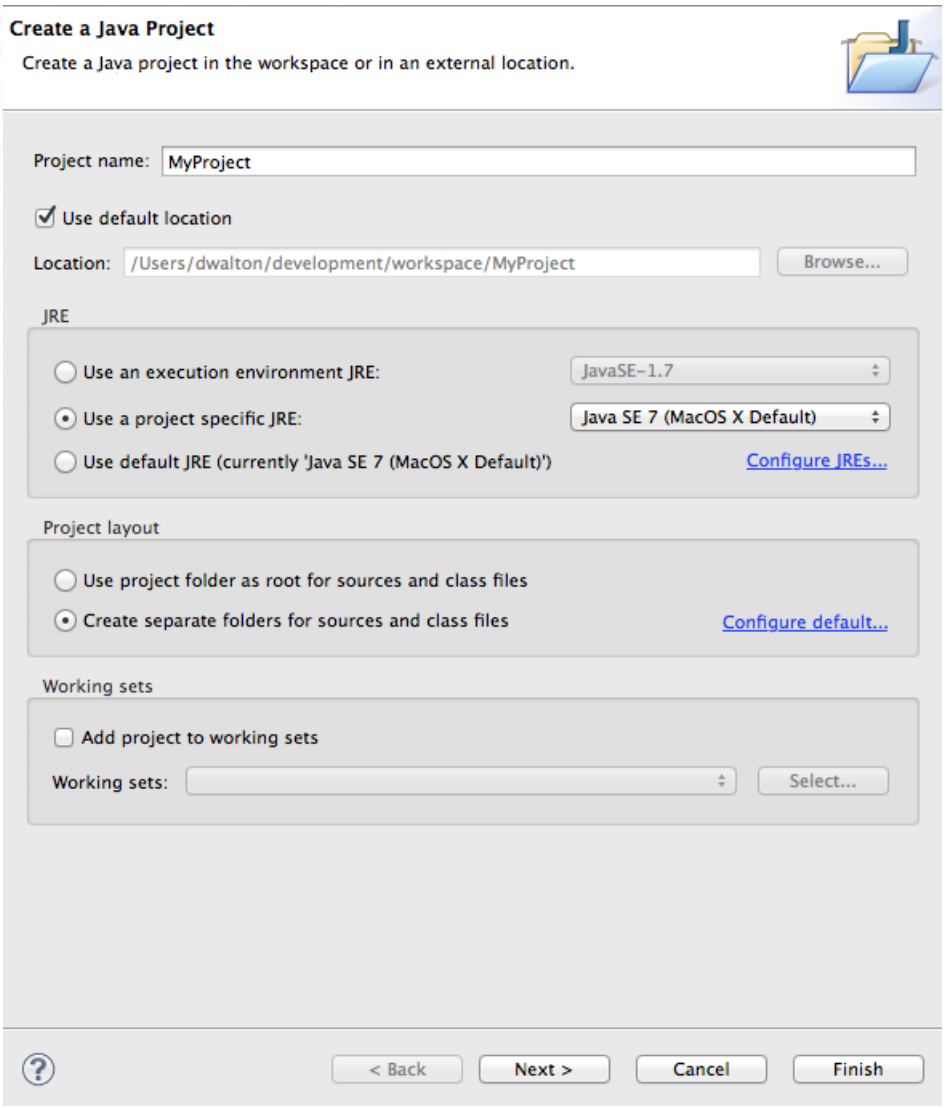


Figure 91: New Java Project

Press **Finish** to create the project. In the **Build Path** add the external jar `lib/diffusion.jar`.

In the `java` folder create the correct package hierarchy to suit your needs, and create the your publisher class that extends the class `com.pushtechnology.diffusion.api.publisher.Publisher`

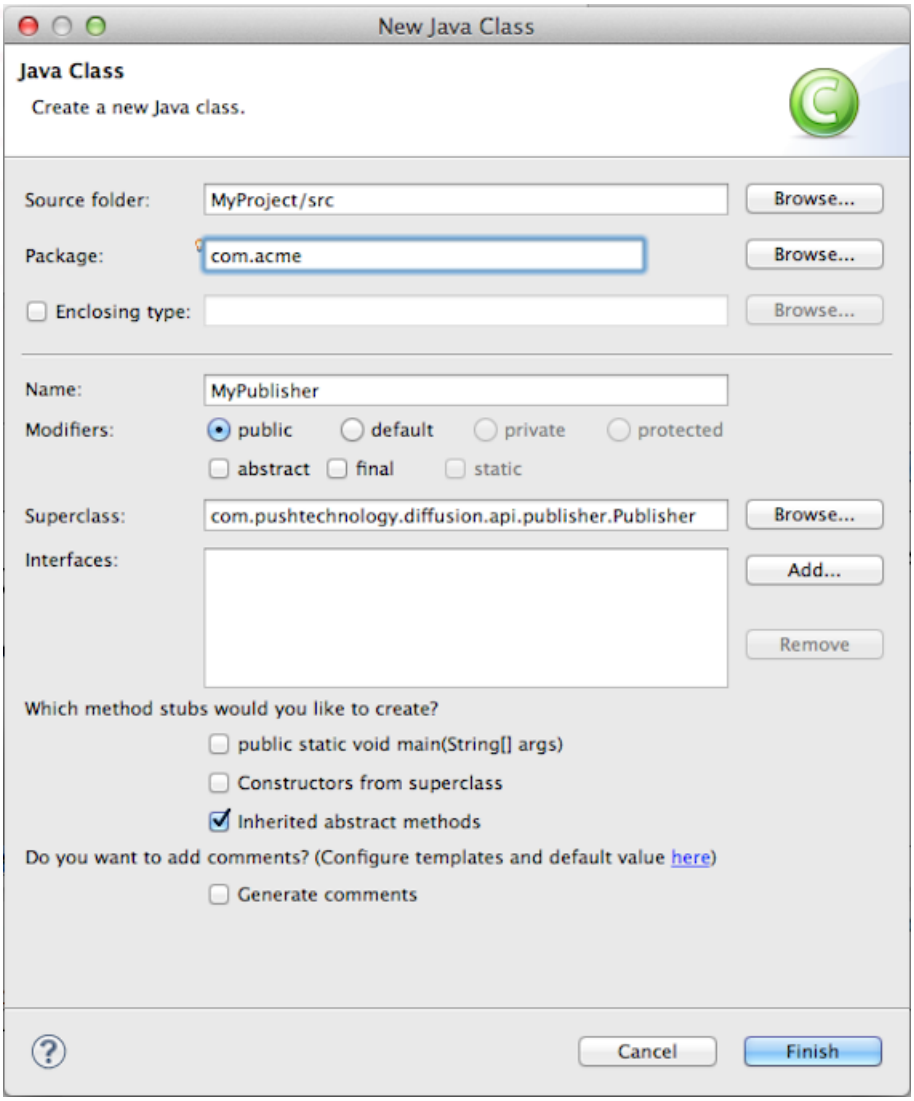


Figure 92: Creating a new Java class

Modify the properties

Modify `etc/Publisher.xml` to load the new publisher. Below is an example of the publisher definition for this publisher.

```
<publisher name="MyPublisher">
 <class>com.pushtechnology.publisher.MyPublisher</class>
</publisher>
```

Create a run target

Open the **Run Configurations** dialog in Eclipse and create a new run configuration, choose `com.pushtechnology.diffusion.Diffusion` as the main task. In the arguments tab, set the working directory to the `bin` directory of the Diffusion installation. Using the classpath tab, add to the user entries all jar files located in your installation's `ext` directory. You can also add any external jar files that are required by your publisher.

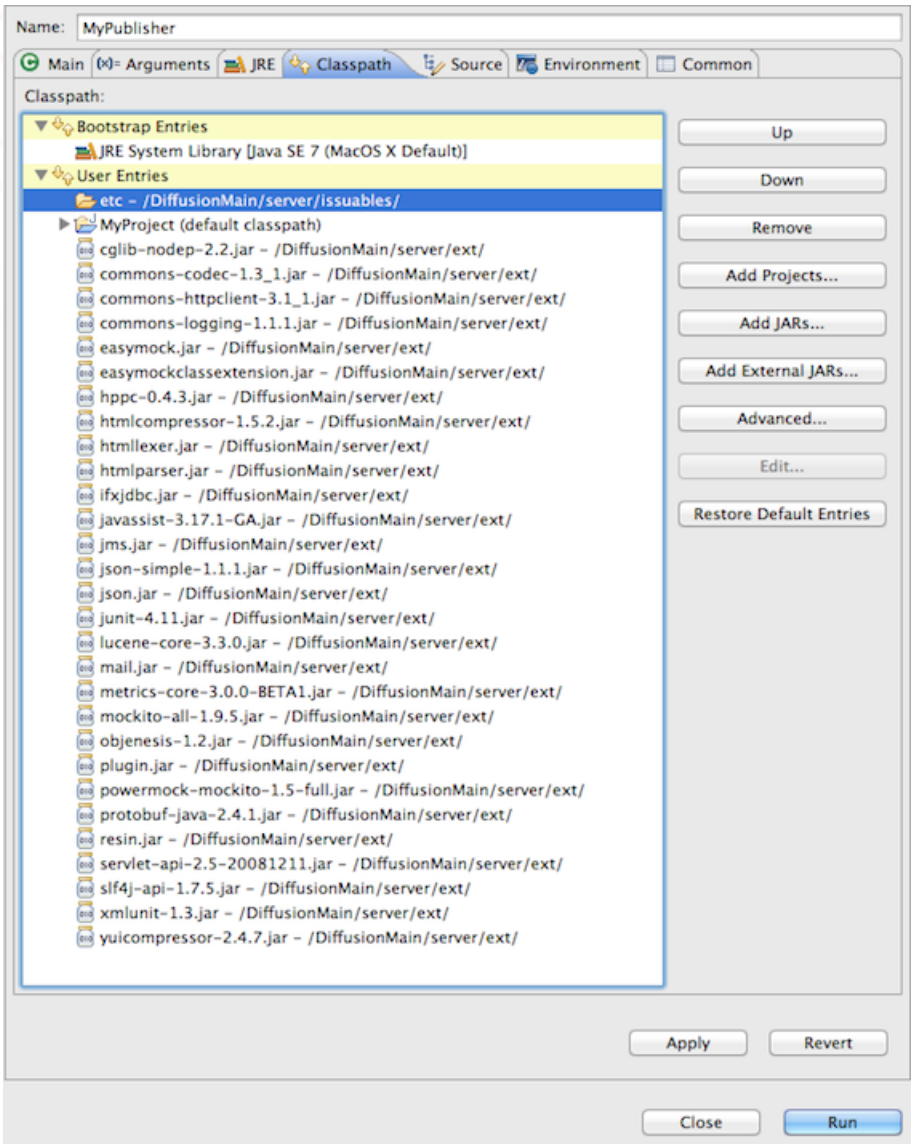


Figure 93: Example classpath entries

You can now set break points within the publisher. By default Eclipse creates a `bin` directory in the `src` directory, which contains the class files for your publisher. By setting the `etc/Server.xml` `usr.lib` entry to this `bin` directory, when you start Diffusion from the command line, your publisher is on the classpath.

Log Messages

Log Reference

Log Messages

All Server Log Messages

Id	Message	Description
PUSH-000001	Attempt #{} to reconnect to JMS Server.	The JMS adapter is currently disconnected from a JMS server and

Id	Message	Description
		has just failed another attempt to reconnect.
PUSH-000002	Cannot stop the JMSAdapter.	Something went wrong during initialization of the JMS adapter, and there was an error when trying to clean up and stop the adapter.
PUSH-000003	Couldn't find JMSReplyTo destination: '{}'.	An attempt has been made to send a message with a JMSReplyTo destination, but the JMS adapter cannot create the ReplyTo destination on the JMS server.
PUSH-000004	Discarding non-TextMessage from JMS: '{}'.	The JMS adapter only supports TextMessage types but it received another message type.
PUSH-000005	Exception from JMS provider '{}'.	The JMS adapter received notification of an exception from the JMS server. Typically, this happens when the connection between the adapter and the server has been terminated.
PUSH-000006	Extra JMS header found, ignoring: '{}'.	JMS messages can have user-defined headers as key/value pairs. Diffusion messages are effectively "value" only. To map user headers from Diffusion to JMS there must be an even number of headers. These headers are artificially grouped as key/value pairs in the JMS message.
PUSH-000007	Failed to find topic binding for '{}'.	An attempt has been made to send a reply from the JMS adapter back to the JMS server using the DiffusionReplyTo header, but the adapter was unable to determine the corresponding JMSReplyTo destination.
PUSH-000008	Failed to generate delta message '{}'.	A message has been received from the JMS server and a delta message needs to be sent, but the update or publish of the topic data associated with the Diffusion topic failed.
PUSH-000009	Failed to generate ITL message '{}'.	An error occurred while generating an ITL message for a client after a message has been received from a JMS queue.
PUSH-000010	Failed to get message content of message to JMS, topic is '{}'.	Failed to read/parse the content of a Diffusion message while translating it to a JMS message for sending.

Id	Message	Description
PUSH-000011	Failed to send message to JMS destination '{}', exception: {}.	An error occurred while sending a message from the JMS adapter to the JMS server.
PUSH-000012	Failed to start JMS Adapter.	The JMS adapter was unable to start.
PUSH-000013	Maximum number '{}' of reconnection attempts reached, JMSAdapter disconnected.	The adapter has exhausted the maximum number of configured connection attempts with the JMS server and has given up.
PUSH-000014	Received TextMessage for unknown destination: '{}'.	A message was received by the JMS adapter from the JMS server, but the adapter was unable to determine which Diffusion topic to map the message to.
PUSH-000015	JMS Server disconnect detected, attempting reconnection.	The JMS server has been disconnected from the JMS adapter and the adapter is attempting to reconnect.
PUSH-000016	JMS Server reconnected after '{}' attempts.	The JMS adapter has managed to (re)connect to the JMS server.
PUSH-000017	Unable to create a subscription to '{}', exception is {}.	The JMS adapter failed to create a subscription to the JMS destination associated with the given topic name.
PUSH-000018	Unable to create new topic '{}'.	The JMS adapter has received a message from the JMS server and needs to create a corresponding Diffusion topic, but was unable to.
PUSH-000019	Unable to extract headers from JMS message: '{}'.	Something went wrong while attempting to convert JMS message headers to Diffusion message headers.
PUSH-000020	Unable to get payload from JMS message '{}'.	The JMS adapter has received a message from the JMS server, but is unable to read the content.
PUSH-000021	Unable to restart JMS Server connection.	After an exception from the JMS server, the JMS adapter attempted to reconnect but was unable to.
PUSH-000022	Exception has been thrown by the ServerAckListener.messageNotAcknowledged method of {}.	The specified listener methods has thrown an exception which has been logged but otherwise ignored.
PUSH-000023	Client Auto Failover failed.	A connection to a server has failed, and the auto failover process has encountered an error.

Id	Message	Description
PUSH-000024	Mime extension '{}' was already mapped to '{}' : overwriting map with '{}'.	A configured Mime extension was already mapped to another value but has now been remapped to a new value. This indicates duplicate mime extension values in the mimes configuration.
PUSH-000025	Connection attempt failed with {} - trying next server.	An attempt to connect to a server has failed but more than one possible server has been specified and so the next is being tried.
PUSH-000026	Requested receive buffer size for HTTP Proxy connection not allocated, requested: {}, allocated {}.	Diffusion tried to set the size of the TCP receive buffer for a socket connection to an HTTP proxy, but the actual number of bytes allocated was different.
PUSH-000027	Requested send buffer size for HTTP Proxy connection not allocated, requested: {}, allocated {}.	Diffusion tried to set the size of the TCP send buffer for a socket connection to an HTTP proxy, but the actual number of bytes allocated was different.
PUSH-000028	An exception has been thrown by the {} method in {}.	An exception has been thrown by an implementation of the specified listener interface. The exception is logged but has no further impact.
PUSH-000035	Unable to send fetch reply to Client '{}'.	An error occurred while sending a fetch reply message back to a client.
PUSH-000036	Failure fetching state from '{}' of '{}'.	While handling a fetch request for a topic with topic data, an error occurred while generating a topic load message.
PUSH-000037	Subscription Handler '{}' specified for Routing Topic '{}' - ignored.	A subscription handler has been specified for a routing topic which is not allowed therefore the handler has been ignored.
PUSH-000038	Exception caught from TopicDeletionListener '{}' topicDeleted method.	An exception has been thrown from a call to a TopicDeletionListener.topicDeleted method. The exception has been logged but has no other effect.
PUSH-000039	Exception caught from TopicTreeListener '{}' {} method.	An exception has been thrown from a call to a TopicTreeListener method. The exception has been logged but has no other effect.
PUSH-000040	WhoIs connection failure limit exceeded - not resolving.	Five attempts to connect to the WhoIs have failed. There will be no WhoIs service.

Id	Message	Description
PUSH-000041	Failure to connect to WhoIs provider at {}:{} - will retry 5 times.	Unable to connect to the WhoIs provider indicated. The connection will be attempted up to five times before giving up.
PUSH-000042	Failed to enumerate files in usr.lib directory '{}'.	An error has occurred while traversing the files within the given directory.
PUSH-000043	Attempted to load duplicate jar file '{}'.	An attempt was made to load the given file more than once.
PUSH-000044	Entry '{}' in server configuration is not a directory.	The given file system name must identify a directory.
PUSH-000045	Attempt to queue message '{}' for client '{}' when ACK timeout has already expired.	An acknowledgment would be queued in response to a message requiring an acknowledgment. However the ACK timeout has already expired. Consider increasing the ACK timeout.
PUSH-000046	Client '{}' closing - {}.	The given client is closing.
PUSH-000047	Client '{}' closing - {}, exception='{}'.	The given client is closing due to an exception.
PUSH-000049	Error subscribing Client '{}' to Topic '{}'.	An error has occurred while trying to subscribe the specified client to the specified topic.
PUSH-000050	Client '{}' sent command message to Topic '{}' which does not have command-based Topic Data. Message is '{}'.	Command messages can only be processed by topics with related command-based topic data.
PUSH-000051	Failed to apply Client change event : '{}'.	An error occurred while applying a change to the given client.
PUSH-000052	Exception in ACK listener '{}' messageNotAcknowledged method.	An error occurred while informing a listener that a dispatched message has not been acknowledged.
PUSH-000053	Failed to queue background close of client '{}'.	An attempt to queue the close of a client has failed and therefore the client has been closed in the current thread.
PUSH-000054	Client '{}' tried to fetch invalid topic selector '{}'.	The topic selector expression supplied by a client for a fetch request was invalid.
PUSH-000055	Client group "'{}'" failed to send message to '{}'.	In sending a message to a client group, transmission to the given individual client encountered an error.
PUSH-000056	User licence limit breached, connection rejected.	The maximum number of licensed connections has been exceeded.

Id	Message	Description
PUSH-000057	Server is not licensed to accept distributed connections.	This Diffusion server is not licensed to accept connection from other Diffusion servers.
PUSH-000058	Server is not licensed to accept mobile connections.	The licence does not allow for Diffusion Mobile-to-Server connections.
PUSH-000060	Unable to subscribe Client '{}' to Topic '{}' - Timed out attempting to acquire lock.	A timeout has occurred while attempting to acquire a lock on a topic in order for a client to subscribe to it.
PUSH-000061	Failed to subscribe Client '{}' to Topic(s) '{}' - Invalid Topic name or Selector.	The client sent an invalid topic selector pattern for subscription.
PUSH-000062	Failed to schedule future close of client '{}'.	It was not possible to schedule the future close of a client that is being kept alive for reconnect purposes.
PUSH-000063	Unable to send fetch reply message to Client '{}' for Topic '{}'.	A failure occurred while trying to send a fetch reply to a client.
PUSH-000064	Failed to unsubscribe Client '{}' from '{}' - Invalid Topic name or selector.	The client sent an invalid topic selector pattern for unsubscription.
PUSH-000065	Failure processing message from Server.	An error has occurred at the client end of a connection to a server while processing a message from the server.
PUSH-000070	Invalid connection from '{}' ('{}').	An invalid connection has been attempted from the given address and location.
PUSH-000072	Connector '{}' created for connections of type {}, listening on {}.	The given connector was created to support the given types of connection.
PUSH-000074	Connector '{}' received an HTTP connection but does not support HTTP.	Connector has received an HTTP connection attempt but does not support HTTP connections. The connector does not define a valid web server.
PUSH-000075	Unable to start Connector '{}'.	An error occurred while trying to start a connector.
PUSH-000076	Unable to start Connector '{}', permission denied.	The operating system denied permission for a resource used by this connector. Often this relates to reserved ports (less than 1024) on Unix-based operating systems.
PUSH-000077	HTTP Method '{}' is not supported in request from '{}'. Request: '{}'.	An HTTP client has requested a method that is not supported by Diffusion.

Id	Message	Description
PUSH-000078	HTTP Processing error, request from '{}'.	An error occurred in the processing of an HTTP request from the given client.
PUSH-000079	Connector '{}' received a Policy File request which is not supported.	Connector has received a policy file request but does not support such requests. The connector does not have a policy file configured.
PUSH-000080	Non-SSL connection attempted to connector '{}' which only supports SSL-type connections.	An attempt has been made to make a non-secure connection to a connector that is configured to only accept secure connections.
PUSH-000081	Connector '{}': no connection - SSL handshake error: {}.	A secure (SSL) connection was attempted but the SSL handshake failed.
PUSH-000082	SSL connection attempted but connector '{}' does not support SSL.	An attempt has been made to make a secure (SSL) connection via a connector that does not support SSL. The connector does not define a keystore.
PUSH-000083	Connector '{}' received a client connection but does not support clients.	The given connector is not configured to receive and handle client connections. For example, the connector might be dedicated to event publisher connections.
PUSH-000085	{}: Requested input buffer size could not be allocated, requested: '{}' allocated: '{}'.	The receive buffer of the socket was assigned a different amount of memory than requested. This is configured by the input buffer size. The configured input buffer size is a hint to the operating system. Refer to your OS documentation for any socket buffer limits. This can have performance implications.
PUSH-000086	{}: Requested output buffer size could not be allocated, requested: '{}' allocated: '{}'.	The send buffer of the socket was assigned a different amount of memory than requested. This is configured by the output buffer size. The configured output buffer size is a hint to the operating system. Refer to your OS documentation for any socket buffer limits. This can have performance implications.
PUSH-000087	Connector '{}' received an unidentified connection request [{}] from {}.	An unidentified connection attempt has been made via a connector. The hexadecimal representation of the initial bytes of the connection request are logged in order to aid in identifying the origin.

Id	Message	Description
PUSH-000089	Failed to update Child List Topic Data for Topic '{}'.	An unforeseen error occurred while updating this ChildListTopicData.
PUSH-000090	An exception has been thrown from CustomTopicDataHandler.{} method.	The custom TopicData implementation threw an exception in the given method.
PUSH-000091	Client '{}' sent invalid topic selector pattern '{}' to Topic Notify Topic.	A client has sent an invalid topic name or selector pattern to a topic notify topic selection.
PUSH-000092	An exception has been thrown from TopicNotifyTopicListener.{} method in '{}'.	The topic notify topic listener threw an exception in the given method.
PUSH-000093	Topic Notify Select Request Mode '{}' from Client '{}' is invalid.	Client has sent an invalid select request mode to a topic notify topic.
PUSH-000094	{} was unable to notify Client '{}' of Topic '{}' notification event - Client closing.	A failure has occurred notifying a client of a topic addition, removal or update - the client will be closed.
PUSH-000095	Unrecognised notification type '{}'.	The notification request holds unexpected content.
PUSH-000096	Client '{}' request to lock paged data '{}' timed out.	Unable to action a request from a given client on the specified data as it was not possible to acquire a lock within the timeout period.
PUSH-000097	Paged topic '{}' Unable to process command '{}' from client '{}'.	A paged topic was unable to process a command from a client.
PUSH-000098	Failure sending a paged topic status message to client '{}'.	A failure occurred while sending a paged topic status message to the specified client.
PUSH-000099	Failure sending a paged topic update message to client '{}'.	A failure occurred while sending a paged topic update message to the specified client.
PUSH-000100	Command topic data '{}': received invalid command message '{}'.	A client has sent an invalid command message to command topic.
PUSH-000142	Error removing routing data {} from {}.	A failure occurred while trying to remove a topic from a routing topic.
PUSH-000144	Routing topic failed to subscribe client '{}' to topic '{}'.	A routing topic failed to subscribe a client to a topic.
PUSH-000145	An exception has been thrown from ServiceTopicListener.{} method in '{}'.	The service topic listener threw an exception in the given method.
PUSH-000146	Service Request error for client '{}' on service '{}', request id '{}', error '{}'.	A request from a client to a service topic has failed and it was not possible to report the failure back to the client, possibly because it is closed.

Id	Message	Description
PUSH-000147	Failure in TopicDataListener '{}' topicDataChanged method.	An exception has been thrown by an implementation of the TopicDataListener.topicDataChanged method. The exception is simply logged and has no other effect.
PUSH-000148	Unable to lock topic {} in order to notify routing topic data instances of topic removal.	A timeout occurred while waiting to lock the specified topic in order to notify instances of RoutingTopicData of the Topic removal. This error might leave one or more instances of RoutingTopicData in an inconsistent state.
PUSH-000150	Message '{}' received from '{}' by '{}' has not been handled.	A command topic that does not handle messages from clients has received a message from a client and ignored it.
PUSH-000151	Unable to get JMX MBean attribute.	It was not possible to retrieve an attribute from a JMX MBean.
PUSH-000152	Unable to load shutdown hook class '{}': {}.	An error occurred loading the given third party class.
PUSH-000153	Unable to load startup hook class '{}': {}.	An error occurred loading the given third party class.
PUSH-000154	Caught JVM shutdown.	The JVM shut down unexpectedly.
PUSH-000155	No connectors have been configured. Creating default connector listening on port {}.	No connectors have been configured therefore a default connector listening on port 8080 has been created with all default values.
PUSH-000156	No Multiplexer Configuration. Creating default configuration.	No multiplexer configuration has been found therefore a default multiplexer configuration will be assumed.
PUSH-000157	No queue definitions configured. Creating DefaultQueue.	No queues have been configured therefore a default queue definition has been assumed.
PUSH-000158	No thread pools configured. Created new pool definition called '{}'.	No thread pools were configured therefore a default one has been added.
PUSH-000159	The maximum message size is {} bytes.	The maximum message size has been established.
PUSH-000160	No inbound pool has been configured - using '{}'.	No inbound thread pool has been configured therefore the first configured thread pool has been assumed to define the inbound pool.

Id	Message	Description
PUSH-000161	Diffusion - Removing Publishers.	The server is exiting, and all publishers within are being unloaded.
PUSH-000162	Running shutdown hook '{}'.	The given third party class is being executed at server shutdown.
PUSH-000163	Running startup hook '{}'.	The given third party class is being executed at server startup.
PUSH-000164	Diffusion Server not started.	Diffusion server failed to start.
PUSH-000165	Diffusion Server started.	Diffusion started successfully.
PUSH-000166	Diffusion Server '{}' starting.	The Diffusion server is starting.
PUSH-000167	Diffusion Stopped.	The Diffusion server has been stopped.
PUSH-000168	Diffusion Stopping, reason='{}' by administrator='{}'.	Diffusion is processing a shutdown request.
PUSH-000169	Diffusion - Stopping Connectors.	Connectors are being stopped during a shutdown of Diffusion.
PUSH-000173	Exception notifying '{}' of '{}'.	An exception occurred while processing an internal asynchronous event.
PUSH-000174	Unable to submit event '{}' to '{}' for execution.	A failure has occurred while submitting a notification event to the specified thread pool for execution.
PUSH-000175	SHA1 algorithm not found, unable to respond to WebSocket request.	The WebSocket protocol requires the SHA1 algorithm, but it was not found. This can only happen with an incomplete or incompatible Java installation.
PUSH-000177	Received invalid message '{}' from '{}': {}.	The server module dealing with Introspector clients has received an invalid message.
PUSH-000178	Exception publishing message on topic '{}'.	The server module dealing with Introspector clients caught an exception sending a message to the given client.
PUSH-000180	Topic '{}' might not be removed by Introspector for client {}.	The given client has attempted to remove a topic that cannot be removed.
PUSH-000181	Topic '{}' not removed for client '{}': {}.	The given topic was not removed for the given client and reason.
PUSH-000182	Topic '{}' removed by Introspector client '{}'.	The given topic was removed for the given client.

Id	Message	Description
PUSH-000183	Message channel error: '{} : {}' caused by '{}'.	A communication error has occurred on a message channel.
PUSH-000184	Connector '{}:' has no more acceptors - stopping.	An acceptor for a connector has failed, and there are no more acceptors available to service connection requests therefore the connector is stopping.
PUSH-000185	Failed to accept connection for acceptor '{}' on connector '{}'.	This can occur when a socket connection has been made to Diffusion, but a failure occurred while initializing it.
PUSH-000187	Second attempt to write output, try increasing timeout or output buffers '{}'.	Diffusion has tried to obtain a selector for writing and failed and is trying again. If this problem persists, try increasing the write timeout or the output buffer size.
PUSH-000188	Selector thread failure {} - Closing.	An error has occurred in a selector thread and the thread is closing.
PUSH-000190	SSL unwrap error: '{}'.	An error occurred while decoding SSL-wrapped data.
PUSH-000191	Connector '{}:' - Unable to accept connection.	An exception occurred while attempting to accept a socket connection.
PUSH-000193	SSL Keystore loaded from '{}'.	SSL keystore has been loaded from the specified location.
PUSH-000194	JMX cannot expose '{}' as a topic as dimensions exceed 2.	Only 1- and 2-dimensional arrays can be mirrored as a topic.
PUSH-000195	JMX: Cannot register object '{}' at MBean ObjectName '{}'.	An error occurred registering the given object with the given JMX ObjectName with the JMX server.
PUSH-000196	'{}:' failed to update Diffusion license.	An error occurred while attempting to update the license file.
PUSH-000197	Unable to create directory '{}' for the public certificates.	The HTTP service which can be used for updating the license needs to be able to store the license in a directory, but that directory does not exist and cannot be created.
PUSH-000198	Licence is invalid.	The license is invalid.
PUSH-000199	License is not valid for this version of Diffusion (license='{}' vs '{}').	The license file does not match this version of Diffusion.
PUSH-000200	Product '{}' not licensed for this address scheme.".	A product is not licensed for either IP address or MAC addressing schemes.
PUSH-000201	Soft user limit ({}) for {} exceeded. Hard limit at {}.	Once the number of connections for a product reaches its soft limit,

Id	Message	Description
		a warning is emitted. The product might cease to function if this number reaches the hard limit.
PUSH-000202	Product license expires in {} day(s).	The installed license file is nearing expiration. A new license will be required soon for Diffusion to continue running.
PUSH-000203	License has expired.	The Diffusion license has expired.
PUSH-000204	License hard user limit ({}) for {} exceeded.	Once the number of client connections has reached or exceeded a the hard limit, the product might be disabled.
PUSH-000205	Licence for Introspector has expired.	Licence for Introspector has expired.
PUSH-000206	Licensor stopping Diffusion. Invalid license.	The Diffusion license is invalid.
PUSH-000207	License check failed : Unable to check server-side addressing for product [IP] '{}'.	During the license check, Diffusion attempts to check the server's IP addresses but it was unable to do so.
PUSH-000208	License check failed : Unable to check server-side addressing for product [MAC] '{}'.	Diffusion is licensed to run on a server with a specific MAC address, but an error occurred while looking for MAC addresses on the server.
PUSH-000209	License check failed : Unable to match MAC address for ['{}'].	The product is licensed for a particular MAC address, but the server does not have this MAC.
PUSH-000210	License check failed : Unable to parse CIDR Address: '{}'.	The product is licensed to run on machines with IP addresses matching a CIDR expression, but the expression could not be parsed. Contact Push Technology for an updated license file.
PUSH-000211	License version is empty, this is not allowed.	License version is empty, this is not allowed. Contact Push Technology for a replacement license file.
PUSH-000212	Log level set to '{} for '{}'.	The logging level of the specified log file has been changed as indicated.
PUSH-000214	Invalid JMX credentials.	The supplied JMX credentials are incorrect.
PUSH-000215	Remote JMX management service is disabled.	The remote JMX service is configured not to start.
PUSH-000216	Remote JMX management service has started. Listening to {}.	The remote JMX service has started.
PUSH-000226	Multiplexer error while processing client '{}'.	A multiplexer tried to send messages for a client, but failed.

Id	Message	Description
PUSH-000227	Multiplexer event is delayed as the queue is full.	A multiplexer cannot queue an event because the queue is full, but is continuing to try to do so.
PUSH-000228	Multiplexer event is significantly delayed as the queue is full.	Because the queue is full, a multiplexer cannot queue an event after a significant amount of time, but it is continuing to try to do so.
PUSH-000229	Error while handling a multiplexer event.	Error while handling a multiplexer event.
PUSH-000230	Failed to schedule event.	A multiplexer tried to schedule an event for some time in the future, but was not able to do so.
PUSH-000231	Possible latency warning from '{}' [{}ms].	The configured latency warning time has been exceeded while a multiplexer is sending to a client.
PUSH-000232	No multiplexers configured.	No multiplexer definitions have been configured so a default has been assumed.
PUSH-000233	Multiplexer '{}' started .	A multiplexer has started.
PUSH-000234	Properties loaded: '{}'.	The specified properties file has been loaded.
PUSH-000235	Properties reloaded: '{}'.	The specified properties file has been reloaded.
PUSH-000236	Cannot establish JMX attribute '{}' for MBean '{}'.	The given MBean/Attribute pair is not supported by Diffusion.
PUSH-000237	Cannot get JMX attribute '{}'.	An error occurred while fetching the value of the given MBean attribute.
PUSH-000238	JMX attribute '{}' for MBean '{}' is not supported: {}.	The given MBean/Attribute pair is not supported by Diffusion.
PUSH-000239	Cannot create topic for MBean {}.	It was not possible to create a topic to mirror the content of the given JMX MBean.
PUSH-000240	JMX Adapter unable to get topic '{}'.	An error occurred in the JMX Adapter while trying to get the given topic from the topic tree.
PUSH-000241	Cannot remove topic for MBean {}.	It was not possible to remove a topic that mirrors the content of the given JMX MBean .
PUSH-000242	JMX configuring property '{}'={}.	The JMX Adapter configuration is being applied.
PUSH-000243	JMX cannot parse property '{}'.	An error occurred reading the given property.

Id	Message	Description
PUSH-000244	JMX exception interacting with MBean '{}'.	An error occurred creating the tree of topics to represent a JMX MBean.
PUSH-000245	JMX exception while updating topic '{}'.	An error occurred while polling an MBean attribute that the given topic reflects.
PUSH-000246	Caught exception removing notification listener from JMX '{}'.	Diffusion is no longer listening to notification from the given MBean, and this has caused a problem.
PUSH-000247	Cannot find MBean '{}' to listen for notifications.	Diffusion is attempting to add a notification to the given MBean, and this has caused a problem.
PUSH-000248	Listener for MBean '{}' not found.	The notification listener being removed from the given MBean is not currently registered with it.
PUSH-000249	Cannot send message given notification object {}.	A JMX notification was detected and serialised as a message, but it was not possible to send the message.
PUSH-000250	Unexpected notification object '{}'.	A JMX notification of an unexpected type was detected.
PUSH-000251	Unexpected notification type '{}'.	MBeanServerNotification received that was neither about an MBean registering or deregistering.
PUSH-000252	Unable to construct topics for DiffusionPublisher Log Handler.	The Diffusion publisher log handler encountered an error trying to create its topics.
PUSH-000253	Exception caught in {} method of authorization handler {}.	An exception has been thrown by the specified method of the user-written authorization handler - the exception has been logged and the requested action rejected.
PUSH-000254	Authorization manager started with authorization handler class '{}'.	Authorization manager has been started with the specified handler class.
PUSH-000255	Publisher {} connected to server {} as {}.	A publisher has successfully connected to another Diffusion server as a client.
PUSH-000256	Deploying publishers compiled for Diffusion version '{}'.	Publishers compiled for the specified version of Diffusion are being deployed.
PUSH-000257	Publisher '{}' did not handle message from server '{}'.	A publisher has received a message from a server but has not provided an implementation for the messageFromServer method.
PUSH-000258	Publisher '{}' did not handle server rejected credentials for '{}'.	The default (unimplemented) serverRejectedCredentials method

Id	Message	Description
		of a publisher has been called because credentials it has sent to a server have been rejected.
PUSH-000260	Diffusion-Version not present in manifest file, not deploying.	The publisher will not be deployed because Diffusion-Version is not present its DAR manifest file.
PUSH-000261	Error deploying publisher from file '{}'.	An exception has occurred while trying to deploy a publisher from a DAR file.
PUSH-000262	Error writing file for deployment: {}.	When hot-deploying publishers, the contents of DAR files must be extracted to disk. Check for write permissions to the deployment directory and that there is sufficient free disk space.
PUSH-000263	Exception caught in publisher '{}' in method '{}'.	A publisher implementation has thrown an exception in the given method. The exception has been logged but otherwise ignored.
PUSH-000264	Exception caught in publisher '{}' in systemStarted method.	Publishers declared in etc/Publishers.xml can be informed that Diffusion is ready by overriding the systemStarted() method. This message is emitted if an exception is thrown by the publisher's systemStarted() method.
PUSH-000265	Failed to load publisher '{}'.	Diffusion was unable to load a publisher, probably because of a problem with the publisher's configuration files.
PUSH-000266	This DAR file is for Diffusion version {} or greater. Not deploying on Diffusion version {}.	The META-INF/MANIFEST.MF of the DAR file contains the "Diffusion-Version" attribute that specifies with which versions of Diffusion it is compatible. The DAR is marked as incompatible with this version of Diffusion and will not be deployed.
PUSH-000267	Introspector publisher defined, but not licensed.	Diffusion is attempting to start the publisher which is required for the Introspector, but the license file does not allow the Introspector to be run.
PUSH-000268	Publisher '{}' is not stoppable, not undeploying.	An attempt has been to undeploy a publisher, but its isStoppable() method returns false. The publisher will not be undeployed.
PUSH-000269	JMX adapter threw exception while starting.	An unexpected exception prevented the JMX adapter from starting.

Id	Message	Description
PUSH-000270	Configured not to poll MBeans.	The JMX adapter is configured not to poll those JMX MBean attributes with a matching topic subscription.
PUSH-000271	JMX adapter configured not to start.	The JMX adapter is configured not to start, and will be unavailable.
PUSH-000272	Polling MBeans every {} milliseconds.	The JMX adapter is configured to configured to poll required MBean attributes for the given frequency.
PUSH-000273	JMX adapter starting.	The JMX adapter has begun construction.
PUSH-000274	DeploymentMonitor: The DAR file '{}' has not been fully undeployed. The publisher(s) {} have not been undeployed. The DAR file will be ignored from now on.	An error occurred during an attempt to undeploy a publisher, which had been deployed using the "hot deploy" feature. This can happen if a DAR contains multiple publishers, but not all of them return "true" from the isStoppable() method.
PUSH-000275	Publisher '{}' received notification of non-acknowledgment of message '{}' by {} clients.	A publisher has received a notification that one or more clients have not acknowledged a message requiring acknowledgment, but the publisher has not implemented the messageNotAcknowledged method to handle it.
PUSH-000282	Removed publisher '{}'.	The given publisher has been removed from the server.
PUSH-000283	Publisher to server connection '{}' closing due to send failure: {} {}.	A connection between a publisher and a Diffusion server is being closed due to a failure while sending messages to the server.
PUSH-000284	Publisher '{}' unable to connect to server '{}' - no recovery policy specified.	A publisher server connection has failed to connect and no recovery policy has been specified.
PUSH-000285	Started publisher '{}'.	A publisher has been started.
PUSH-000286	Starting publisher '{}'.	A publisher is starting.
PUSH-000287	Failed to start statistics handler for publisher '{}'.	A failure has occurred while trying to start the statistics handler for the specified publisher.
PUSH-000288	Stopped publisher '{}'.	The given publisher has been stopped, and can be restarted.
PUSH-000289	Stopping Diffusion server due to stop-server-if-not-loaded flag for publisher '{}'.	Diffusion failed to start a publisher which has the "stop-server-if-not-loaded" configuration value set to "true" in Publishers.xml. Diffusion will now be stopped.

Id	Message	Description
PUSH-000290	Failure fetching state of {} from publisher {}.	While handling a fetch request for a topic with no topic data, an error occurred while generating a topic load message.
PUSH-000291	Publisher server connection '{}' message queue limit reached.	The specified publisher server connection is being closed because the number of messages in its outbound queue has reached the configured limit.
PUSH-000292	Publisher server connection {} encountered an unexpected error while sending - closing.	A publisher server connection encountered an unexpected error while sending and will be closed.
PUSH-000293	Publisher '{}' unable to connect to server '{}' - will retry every {} milliseconds.	A publisher server connection cannot be established. The recovery policy indicates that the connection be retried and so an automatic retry will occur at the interval specified.
PUSH-000294	Failed to read DAR file '{}'.	Diffusion tried to deploy a publisher from a DAR file, but the DAR was unreadable. Check file permissions and for corruption with the "jar" tool.
PUSH-000295	Publisher '{}' not currently deployed, cannot undeploy.	An attempt was made to undeploy a publisher which is not deployed. Either check that this is the case, or that you have the correct spelling of the publisher's name.
PUSH-000296	DeploymentMonitor: One or more of the publishers in '{}' cannot be stopped. Not undeploying file. The DAR file will be ignored from now on.	An attempt was made to undeploy a hot-deployed publisher, but that publisher does not return "true" from its isStoppable() method.
PUSH-000298	No queue definitions configured : Adding DefaultQueue.	No queue definitions have been configured therefore a default queue configuration is being assumed.
PUSH-000299	A conflation merge error has occurred in '{}'.	An instance of MessageMerger has thrown an exception from its merge method. The exception is logged and no conflation occurs.
PUSH-000300	Message of size {} is too large for {} which has a limit of {}.	An attempt has been made to send a message to a client that is larger than it is possible to transmit to that client, possibly because of buffer limitations.
PUSH-000301	Default queue definition {} does not exist. '{}' used.	There is no configured queue definition with the name as specified as the default queue

Id	Message	Description
		therefore the first configured queue is assumed to be the default.
PUSH-000302	Default queue definition not configured. '{}' used.	No default queue definition has been specified in the configuration therefore the first configured queue definition has been assumed to be the default.
PUSH-000303	Queue definition '{}' not known - using default.	The specified queue definition is not known within the configuration therefore the default queue definition has been assumed.
PUSH-000306	Deployment service failed to process HTTP request.	The publisher deployment service failed to return an HTTP response to the deploying client.
PUSH-000307	PHP service failed to send 404 response.	An error occurred in the PHP service while sending a HTTP 404 response to the web client.
PUSH-000308	Failed to process file.	The PHP service was unable to process a PHP file.
PUSH-000309	REST Failed to process service request.	REST service failed to process a request.
PUSH-000310	Method '{}' from Diffusion servlet called, not being processed.	The Diffusion servlet's implementation of the given method does nothing.
PUSH-000311	Diffusion servlet: {}.	Indicates the startup status of the Diffusion servlet.
PUSH-000312	Client statistics log name '{}' is not configured.	The log named in the client statistics configuration does not exist in the logs configuration.
PUSH-000313	Client statistics enabled but no log name supplied.	No log name has been specified in the client statistics configuration.
PUSH-000316	Failed to update statistics file '{}'.	It was not possible to write the given statistics file.
PUSH-000317	Statistics manager failure.	An unexpected error occurred while starting the statistics manager.
PUSH-000318	Registering statistics reporter '{}'.	A configured statistics reporter of the given name is about to be registered.
PUSH-000319	Failed to register statistics reporter '{}'.	An attempt to register the named configured statistics reporter has failed - the reporter has therefore been ignored.
PUSH-000320	Started statistics reporter '{}'.	The named statistics reporter task has started.

Id	Message	Description
PUSH-000321	Statistics service failed to load statistics reporter '{}'.	A failure has occurred while attempting to load a configured statistics reporter.
PUSH-000322	Error starting statistics service.	An error has occurred while starting the statistics service.
PUSH-000323	Background thread pool caught exception executing '{}'.	A task being executed by the background thread pool has thrown an exception.
PUSH-000324	Thread pool '{}' could not create new thread.	A thread pool encountered an error while attempting to create a new thread.
PUSH-000325	Thread pool '{}' could not create new thread.	The specified thread pool failed to create an additional thread.
PUSH-000326	Thread pool {} execution of {} failed.	Failure executing a task in the specified thread pool.
PUSH-000327	Thread pool {} lower limit notification failed: {}.	The specified thread pool was unable to schedule the notification of the lower threshold limit being reached.
PUSH-000328	Thread pool {} notification of {} to {} failed.	The execution of a notification to a ThreadPoolNotificationHandler failed.
PUSH-000329	Thread pool notification handler {} loaded for thread pool {} with upper threshold of {} and lower threshold of {}.	The specified notification handler has been successfully loaded for the given thread pool.
PUSH-000331	Thread pool '{}' queue full - aborting execution of '{}'.	Specified thread pool queue has become full and the rejection policy is to abort the task. Increase either the pool maximum size or the queue size.
PUSH-000332	Thread pool {} queue full - task running in current thread.	Specified thread pool queue has become full and the rejection policy is to run the task in the current thread, which might be inefficient. Increase either the pool maximum size or the queue size.
PUSH-000333	Thread pool {} was unable to schedule the execution of notification of rejected execution : {}.	The specified thread pool failed to schedule the notification of a rejected task.
PUSH-000334	Thread Pool {} upper threshold notification failed: {}.	The specified thread pool failed to schedule notification of upper threshold reached.

Id	Message	Description
PUSH-000335	Uncaught exception in thread '{}'.	An uncaught exception has been thrown from a specified thread and logged.
PUSH-000336	Algorithm '{}' not found.	The given cryptographic module could not be found.
PUSH-000337	File service: [{}] adding virtual hosts {} and mapping to {}.	A virtual host has been added to the web server's file service.
PUSH-000338	Unable to process alias entry in file '{}': {}.	Diffusion tried to add a new alias to the web server, but failed.
PUSH-000339	HTMLTopicDataSegment: unable to process topic data for {}.	The web server tried to include the contents of a topic's TopicData into the response, but a failure occurred.
PUSH-000340	HTMLIncludeSegment: unable to populate buffer [{}].	The web server tried to include the contents of another file in the response, but a failure occurred writing to a buffer.
PUSH-000341	Missing attribute '{}' from DiffusionTag {} [{}].	The given DiffusionTag incorrect omits the given attribute.
PUSH-000342	Topic '{}' has '{}' TopicData instead of single value TopicData. DiffusionTag (TopicData) [{}].	The given topic lacks SingleValueTopicData, making it impossible to insert here.
PUSH-000343	Topic {} does not have topic data. DiffusionTag (TopicData) [{}].	The given topic lacks TopicData and therefore any state, making it impossible to insert here.
PUSH-000344	Cannot find file attribute in DiffusionTag [{}].	The given DiffusionTag incorrectly omits the 'file' attribute.
PUSH-000345	Cannot load include file '{}' from DiffusionInclude '{}'.	An error occurred loading the given file.
PUSH-000346	Cannot read specified file for DiffusionTag '{}' [{}].	An error occurred loading the given file specified in the DiffusionTag.
PUSH-000347	Unknown publisher in DiffusionTag {} [{}].	The publisher named in this DiffusionTag is not loaded, or does not exist.
PUSH-000348	Unknown topic for DiffusionTag (TopicData) {} [{}].	The topic named in this DiffusionTag does not exist.
PUSH-000349	Unknown type attribute '{}' for DiffusionTag [{}].	The type attribute of this DiffusionTag holds an unexpected value.
PUSH-000350	Web server service cannot process HTML tag '{}' for the publisher '{}'.	An error occurred while processing an HTML embedded tag.
PUSH-000351	{}: Connection rejected as request was missing the HTTP header field '{}'.	This connection was rejected as it was missing the given HTTP header. The connection is likely not from a Diffusion client, or

Id	Message	Description
		an intermediary (firewall/load-balancer) has removed the header.
PUSH-000352	{}: connection rejected as [{}] did not match '{}'.	The connection from an untrusted host was rejected, in accordance with CORS configuration.
PUSH-000353	{}: Connection rejected as [{}] did not match {}.	The connection was rejected because its WebSocket origin did not match the regular expression stored in the configuration.
PUSH-000354	{}: cors-origin not defined or invalid. Aborting request.	CORS (Cross Origin Resource Sharing) request has been received by the web server client service but no "cors-origin" has been configured or the configured value was invalid. The request has been aborted.
PUSH-000355	{} unable to process HTTP Request '{}'.	Web server file service was unable to process the given HTTP request.
PUSH-000356	Client service {}: HTTP processing error on connector '{}': {}.	An error occurred handling an HTTP request.
PUSH-000357	{}: Invalid regular expression '{}' for CORS origin '{}' - feature disabled.	An invalid regular expression was given for the CORS origin configuration.
PUSH-000358	{}: Invalid regular expression '{}' for WebSocket origin '{}' - feature disabled.	An invalid regular expression was given for the WebSocket origin configuration.
PUSH-000359	Poll request for client '{}' from '{}' failed because no matching client session can be found. The client session might have been closed previously. Poll request will be ignored.	The server has received a poll request for an unknown client session. This might be because the client has been closed by the server, or indicate an incorrectly configured load balancer routing an HTTP connection to a server that was not hosting the session. Consider enabling session-stickiness for HTTP clients.
PUSH-000360	FileService [{}]: not adding virtual hosts '{}' mapping to '{}' as it is not a directory.	Failed to add a virtual host to the web server's file service because the configured mapping does not point to a directory.
PUSH-000361	Unable to service HTTP request for service '{}'.	An HTTP service was invoked, but it failed to process the incoming request.
PUSH-000362	Started HTTP service for '{}'.	An HTTP service was successfully started.

Id	Message	Description
PUSH-000363	{} Unable to create log.	An HTTP service has its own log file defined, but Diffusion was unable to create the logger.
PUSH-000364	Web server {} failed to instantiate HTTP service '{}' : {}.	A web server failed to instantiate the named configured HTTP service.
PUSH-000365	Web server failed to start cache service for virtual host '{}'.	An error occurred while starting the cache service for the given virtual host.
PUSH-000366	Virtual host HTTP request {} is invalid.	An HTTP request being handled by a virtual host is trying to break out of the virtual root.
PUSH-000367	Virtual Host HTTP request {} - unable to read file {}.	A failure has occurred trying to read the specified HTML file.
PUSH-000368	{}: websocket-origin not defined or invalid. Aborting request.	Web Server Client Service has received a Web Socket request but "websocket-origin" has not been configured or is invalid. The request has been aborted.
PUSH-000369	Failed to schedule a periodic tidy of the WhoIs cache.	A failure has occurred while trying to schedule a periodic tidy of the WhoIs cache.
PUSH-000370	WhoIs Provider class instantiation failure calling '{}'.	Unable to instantiate the configured WhoIs provider. Will use the default WhoIs provider.
PUSH-000372	Unable to load GeoIP Database '{}'.	An error occurred while loading or initializing the GeoIP database.
PUSH-000373	WhoIs Service failed to lookup address '{}'.	A failure has occurred in the WhoIs service when trying to look up the given address.
PUSH-000374	WhoIs service starting with {} thread(s).	The WhoIs service is starting.
PUSH-000375	Deprecated connector type RTMP found in Connectors.xml and will be ignored.	Deprecated connector type RTMP found in Connectors.xml and will be ignored.
PUSH-000376	Deprecated element '{}' found in {}.xml.	A deprecated element has been found in specified properties file and should be removed.
PUSH-000377	Failed to load XML properties '{}'.	Failed to load XML properties.
PUSH-000378	Failed to load publisher '{}' configuration from XML properties.	Diffusion was unable to load the configuration for the named publisher from an XML properties file.
PUSH-000379	Failed to load subscription validation policy '{}' from file '{}' : {}.	A failure has occurred loading an XML subscription validation policy file.

Id	Message	Description
PUSH-000380	Failed to parse {} at line {}, column {}: {}.	An XML validation event has occurred while validating an XML property file.
PUSH-000381	Loaded XML {} properties from '{}'.	The specified XML properties have been loaded from the file indicated.
PUSH-000382	Unable to load publisher configuration file {}.	Attempted to load a publisher's configuration file (Publishers.xml), but it cannot be found or is unreadable.
PUSH-000383	Unable to check server-side addressing for product {} [{}] : {}.	An error occurred while attempting to check server-side addressing in the licence file.
PUSH-000384	'{}': Failed to decode licence key '{}'.	Unable to decode licence key.
PUSH-000385	Failed to load subscription validation topic mapping for publisher {} topic {} from file {} : {}.	A failure has occurred loading a publisher topic mapping for a subscription validation policy.
PUSH-000386	Failed to load connection validation policy {} from {}.	A failure has occurred loading XML connection validation policy from the specified file.
PUSH-000387	License check failed to match MAC address for '{}'.	Unable to match a MAC address for the specified product.
PUSH-000388	Failed to save XML properties '{}'.	A failure has occurred while saving XML properties.
PUSH-000389	Server connection '{}' has been added for publisher '{}'.	A configured server connection has been successfully added for a publisher.
PUSH-000390	Failed to add server connection '{}' for publisher '{}'.	SSL keystore file has successfully been loaded for a publisher to server connection.
PUSH-000391	Server connection '{}' for pPublisher '{}' loaded SSL keystore '{}'.	A publisher has created a server connection that has loaded a keystore to establish the SSL context. The keystore is used to securely store key, certificate, and trust information.
PUSH-000392	Publisher '{}' has lost connection to server '{}' - publisher closing.	A publisher has lost connection to a server and the recovery policy indicates that the publisher must be closed.
PUSH-000393	Publisher '{}' has lost connection to server '{}' - no recovery policy specified.	A publisher has lost connection to a server and no recovery policy has been specified.
PUSH-000394	Publisher '{}' has lost connection to server '{}' - will retry every {} milliseconds.	A publisher has lost connection to a server and the recovery policy is to

Id	Message	Description
		retry therefore a retry will occur at the specified interval.
PUSH-000395	Publisher '{}' unable to schedule retry connection to '{}' - publisher stopping.	A failure has occurred attempting to schedule a connection retry between a publisher and a server. The publisher will be closed.
PUSH-000396	Unable to perform substitution within property value '{}' due to syntax error.	A syntax error has been detected while trying to perform environment variable substitution on a configuration property.
PUSH-000397	Deploying publishers in DAR file '{}'.	Deploying the publishers in a single DAR file. There can be multiple publishers in a single DAR file. The publishers will be extracted from the DAR file and the additional configuration files loaded.
PUSH-000399	Directory '{}' does not exist or is not a directory. Using '{}' as a log directory.	The directory of a log definition or the default log directory does not exist or it is not a directory. In this case the temporary file directory is used instead. This is specified by 'java.io.tmpdir'. If the intended directory does exist the problem might be that the relative path is incorrect because the working directory is not what you expect. The simplest way to resolve this is to use absolute file paths to reference log directories. This message might be shown when the log is created and when checking if it needs to be rotated.
PUSH-000400	Unable to add listener when reconnecting to {}.	While reconnecting, the JMS adapter was unable to add a listener for a JMS destination.
PUSH-000401	Unable to create consumer when reconnecting to {}.	While reconnecting, the JMS adapter was unable to recreate a JMS consumer.
PUSH-000402	Unable to create session when reconnecting to {}.	While reconnecting and re-establishing the existing JMS consumers, the JMS adapter failed to create a JMS session.
PUSH-000404	'{}': No poll has been opened by client within poll timeout.	HTTP-based connections require an open polling connection to receive messages. If the client has no open polling connection for a fixed period, the client will be closed as unresponsive.

Id	Message	Description
PUSH-000406	Application handler threw exception {}. [cid={}].	An application handler threw an exception when called.
PUSH-000407	An unexpected response was received from a peer and ignored. [cid={} response={}].	A response for a different conversation set has been received from a peer and will be ignored. Typically this indicates a response has been delivered late, after communication with the peer has been re-established.
PUSH-000408	A response was received from a peer for a finished conversation and ignored. [cid={} response={}].	A response with an unknown conversation ID has been received from a peer. Typically this indicates the conversation has timed out.
PUSH-000409	Unable to close socket channel '{}'.	The socket channel failed to close.
PUSH-000410	Failed to queue background close of the publisher server connection '{}'.	An attempt to queue the close of the publisher server connection failed, therefore the connection has been closed in the current thread.
PUSH-000411	Multiplexer undefined. The PublisherServerConnection '{}' might not be connected.	The publisher server connection attempted to send with a null multiplexer. This might be due to the PublisherServerConnection not being connected.
PUSH-000412	Executor {}: task '{}' finished abnormally.	A background task has thrown an exception. See the log for more information.
PUSH-000413	Received request from client '{}' for the details of client session '{}'.	The server has received a client request for details of a client session.
PUSH-000414	Received request from client '{}' for the details of client session '{}'. The client session does not exist.	The server has received a client request for details of an unknown client session.
PUSH-000415	Received request from client '{}' to close client '{}'.	The server has received a request from a client to close a client session.
PUSH-000416	Received request from client '{}' to close client '{}'. The client session does not exist.	The server has received a request from a client to close an unknown client session.
PUSH-000417	Fetch operation {} for {} was aborted due to missing responses for the following topics: {}.	A fetch operation failed, possibly because of an issue at the publisher or state providing client. Look for earlier messages in the server log.
PUSH-000418	Client provided fetch content for topic '{}' and client '{}', but this server failed to create a fetch response message.	A client provided a response for a topic fetch request from the client, but the response could not be converted to a message.

Id	Message	Description
PUSH-000419	One or more clients have registered to provide a fetch response for topic '{}', but none was available for request from client '{}'.	No client was available to handle the request, or one was chosen and failed while processing. The client fetch handler will be discarded.
PUSH-000420	One or more clients have registered to handle queue events, but none was available for a {} event for client '{}'.	No client was available to handle the queue event, or one was chosen and failed while processing.
PUSH-000421	Received request from client '{}' to set conflation for client '{}' to '{}'.	The server has received a request from a client to change the conflation setting for a client queue.
PUSH-000422	Received request from client '{}' to set conflation for client '{}' to '{}'. The client session does not exist.	The server has received a request from a client to change the conflation setting for an unknown client session.
PUSH-000423	Received request from client '{}' to throttle message queue for client '{}' using the {} policy, limit={}.	The server has received a request from a client to change the throttling policy of a client session.
PUSH-000424	Received request from client '{}' to throttle message queue for client '{}' using the {} policy, limit={}. The client session does not exist.	The server has received a request from a client to change the throttling policy of an unknown client session.
PUSH-000426	{}: command service call failed, reporting error to {}: {}.	An internal service call has failed. An error will be reported to the client session listener.
PUSH-000428	'{}': Has rejected an attempt to connect from a Unified API client. Unified API clients are not supported by this connector.	The connector has been configured to accept connections only of certain client types. An attempt was made to connect by a client of an unsupported type. Either the connector is configured for the wrong type, the client should have connected to a different connector, or an attempt was made to connect a client that has greater functionality than you allow on this connector.
PUSH-000429	'{}': Has rejected an attempt to connect from a Classic API client. Classic API clients are not supported by this connector.	The connector has been configured to accept connections only of certain client types. An attempt was made to connect by a client of an unsupported type. Either the connector is configured for the wrong type or the client should have connected to a different connector.
PUSH-000430	There are currently no controllers for {}.	A control binding has been established, but there are no available controllers.

Id	Message	Description
PUSH-000431	Rejected attempt by client {} to bind a control handler: {}.	This might indicate that another control group has established a binding, or that the client already has a binding.
PUSH-000432	Attempt by client {} to register or unregister from an unknown control point: {}.	This is unexpected and possibly indicates the server and client versions are incompatible.
PUSH-000433	Application handler threw exception {} when called with a response. [cid={} response={}].	An application handler threw an exception when called.
PUSH-000434	Diffusion finished deploying {} components.	Diffusion deployments completed.
PUSH-000435	Task failure in selector thread '{}'.	A task failed to be executed in a selector thread.
PUSH-000436	Unable to create file handler for logger '{}'. Log messages will still appear in the console.	A file handler was unable to be created for the requested logger. Messages will continue to be logged to the console, but will not be recorded to their own file.
PUSH-000437	A message has been received with an unknown topic alias '{}'. It cannot be parsed.	When topic aliasing is enabled a short alias is used to replace topic names. This reduces the size of messages sent. A client still needs to know the topic name so the topic name and alias are sent together in the topic load. If a client does not receive a topic load it is unable to handle the deltas with only the alias. Diffusion assumes that a client will receive a topic load message if a load message is created for that topic. If the message is not sent, this error message will be logged.
PUSH-000439	The authentication manager failed to start because an authentication handler of class {} could not be created.	An instance of a configured authentication handler could not be created.
PUSH-000440	Authentication handler {} threw an exception, denying authentication to {}.	A configured authentication handler failed to process an authentication request.
PUSH-000441	The authorization manager failed to start because an authorization handler of class {} cannot be created.	An instance of a configured authorization handler cannot be created.
PUSH-000442	TopicFetchHandler for topic '{}' returned null, indicating an asynchronous response. This is not supported by the client '{}'.	The client does not support asynchronous fetch responses from a TopicFetchHandler.

Id	Message	Description
	The fetch response (if any) will be discarded.	
PUSH-000443	Publisher '{}' returned null for fetch of topic '{}', indicating an asynchronous response. This is not supported by the client '{}'. The fetch response (if any) will be discarded.	The client does not support asynchronous fetch responses from a publisher.
PUSH-000444	Unable to read aliases file '{}'.	The configured aliases file could not be read.
PUSH-000445	Discarding WhoIs request for {} due to backlog. There are currently {} pending requests.	A WhoIs request has been discarded because there is currently a large number of pending requests. The WhoIsListener will be notified with basic locale information based on the caller's IP.
PUSH-000446	Unable to read subscription polices from '{}'.	The configured subscription policy file could not be read.
PUSH-000447	Failed to load publisher '{}'.	Diffusion was unable to register a publisher, probably because of a problem with the publisher's definition.
PUSH-000448	An error occurred registering a handler with parameters '{}': {}.	An internal error occurred registering a handler.
PUSH-000450	An error occurred unregistering a handler with parameters '{}': {}.	An internal error occurred unregistering a handler.
PUSH-000452	A topic update failed because no updater could be found.	A topic update failed because the client has unregistered the updater.
PUSH-000453	A topic update failed unexpectedly: {}.	A topic update failed. See the log message for further information.
PUSH-000454	Server requested unsupported service {}.	The server has requested an internal service that the client has not registered.
PUSH-000455	Client requested unsupported service {}.	The client has requested an internal service that the server does not support.
PUSH-000456	Notifying session {} of error '{}'.	A session error has occurred. The session error handler will be called.
PUSH-000457	More multiplexers configured than available CPU cores ({} vs {}).	Assigning more multiplexers than there are available CPU cores might lead to degraded performance.
PUSH-000458	Application classpath entry '{}' in is not a directory. Check user libraries in the server configuration.	The names in the user library configuration must identify directories.

Id	Message	Description
PUSH-000459	The configured auto-deployment directory '{}' does not exist.	The configured auto-deployment directory '{}' does not exist.
PUSH-000460	Class loader failed to enumerate files in directory '{}'.	An error occurred reading the files within the given directory.
PUSH-000461	Unable to add file '{}' to the class loader.	A server class loader failed to load the given file.
PUSH-000462	Multiplexer '{}' overflowed before start up - event discarded.	A multiplexer queue overflowed before the multiplexer was started.
PUSH-000464	Update failed for topic '{}'.	An error was encountered while attempting to update a topic.
PUSH-000465	Connector '{}' has pings disabled. Unresponsive HTTP clients will not be detected.	The server relies on pinging in order to detect unresponsive HTTP clients. Unresponsive clients might go unnoticed if the ping frequency is not set.
PUSH-000466	Message stream '{}' threw exception.	A message stream callback raised an exception when called.
PUSH-000467	Unable to establish a tunnel through the proxy at {}:{}.	There was a problem in establishing a tunnel through the specified proxy. There might be an issue in reading from or writing to the proxy.
PUSH-000468	Topic stream '{}' threw exception.	A topic stream callback raised an exception when called.
PUSH-000469	Failure to forward message from {} to client {} for topic {}.	The server failed to forward a message from a client to another client. The message might have been delivered. See the log for further detail.
PUSH-000470	The server failed to forward a message from {} to unknown client {} for topic {}.	The server failed to forward a message as the target client session does not exist.
PUSH-000471	The server failed to forward a message from {} to classic API client {} for topic {}.	The server failed to forward a message to a classic API client. See the log for further detail.
PUSH-000472	Service call to server from session {} failed: {}.	The server rejected a service call due to an error. See the log message for further details.
PUSH-000473	Service call to client {} failed: {}.	The client rejected a service call due to an error. See the log message for further details.
PUSH-000474	Call to the registered missing topic handler '{}' failed.	A request made by the server to a missing topic handler resulted in an error.
PUSH-000475	The '{}' secret key encryption algorithm is not available.	The specified password encryption algorithm is not available which

Id	Message	Description
		means that a less secure default mode of password encryption will be used.
PUSH-000476	No authentication handlers are configured. All sessions will be anonymous.	No authentication handlers are configured. All sessions will be anonymous.
PUSH-000477	Configurations that specify only control authentication handlers are invalid.	If a control authentication handler is configured, you must also configure a local authentication handler (such as the system authentication handler) to allow a client that registers a control authentication handler to connect.
PUSH-000478	"{}" at line {} column {}.	An error occurred while parsing security store language commands at the indicated line and column number.
PUSH-000479	Recovery of {} after a write error failed - store must be manually recovered from backup.	Writing of a store file failed and automatic recovery from backup was attempted but also failed. In this event the store has been disabled and must be recovered manually from the backup file.
PUSH-000480	Recovery of "{}" required as backup "{}" exists.	A store backup file has been found which suggests that recovery of the named store might be required. Inspect the store file and replace it with the backup if necessary, and then delete the backup.
PUSH-000481	Errors detected while parsing persistent store file "{}".	Errors were detected while parsing the specified persistent store file - a detailed list of the errors will follow in the log.
PUSH-000482	Error reading persistent store file "{}".	A read error has occurred on a security persistence file.
PUSH-000483	Failure writing to the {} persistence store.	An error has occurred while writing to the named persistence store. The store file could be corrupt and might need to be recovered from a backup.
PUSH-000484	Failure to forward message from {} to {} for topic {}.	The server failed to forward a message from a client to a client that handles messages for the given topic path. The message might have been delivered. See the log for further information.

Id	Message	Description
PUSH-000485	{}: backlog during initialisation - {} queued events sent so far, {} remain on queue.	A session details dispatcher is taking a while to deliver the backlog of pending events to the control client session.
PUSH-000486	{}: closed.	A session details dispatcher was closed.
PUSH-000487	{}: draining queued updates.	A session details dispatcher is starting. It is about to process a backlog of pending events.
PUSH-000488	{}: initialised.	A session details dispatcher has completed initialisation.
PUSH-000489	{}: initialising. Sending {} initial client notifications.	A session details dispatcher is starting.
PUSH-000490	Session error handler {} threw an exception.	An application session error handler threw an exception when called.
PUSH-000491	Session listener {} threw an exception.	An application session listener threw an exception when called.
PUSH-000492	Failed to register topic update source for path {} by session {}.	The server rejected a request to register an update source.
PUSH-000493	WhoIs Service failed unexpectedly.	The WhoIs Service failed.
PUSH-000494	Failed to execute will {} for session {} because of insufficient permissions.	A client session will to remove a topic could not be executed because the session lacks the required permissions.
PUSH-000495	Failed to execute will {} for session {} as the topic does not exist.	A client session will to remove a topic cannot be executed because the topic does not exist.
PUSH-000496	Unexpected error while sending message.	An error occurred when trying to send a message.
PUSH-000497	Connector '{}' does not support Clients.	A connector has received an attempted connection which indicates that it is a Diffusion client but the connector does not support clinet connections.
PUSH-000498	Connector '{}' has received data on {} that has not come from a proxy.	The connector has been configured to require all connections use a proxy protocol but a connection has been made that has not provided proxy information.
PUSH-000499	Failed to generate snapshot for topic {}.	The attempt to generate a snapshot for distribution through replication failed due to topic locking issues. You should ensure that no topic lock is held and not released and attempt to reduce contention.

Id	Message	Description
PUSH-000500	Introspector client '{}' has insufficient permissions for '{}' request.	The specified Introspector client does not have sufficient permissions to perform the request.
PUSH-000501	Diffusion publisher failed to publish a log message to internal topics, disabling handler.	The Diffusion publisher failed to publish a log message. The log handler has been disabled.
PUSH-000502	Using temporary buffer for {}. Formatted message requires {} bytes and configured output buffer size of {} bytes is insufficient.	The configured buffer size was too small to accommodate a message. Consider changing the configuration.
PUSH-000503	An blocking operation failed because the multiplexers failed to process it within {} milliseconds.	This indicates that the server is severely overloaded or deadlocked.
PUSH-000504	No control handler found for routing topic '{}', discarding subscription for client '{}'.	A routing handler must be registered for routing subscriptions to complete.
PUSH-000505	Unable to lock topic {} in order to remove slave instances.	A timeout occurred while waiting to lock the specified topic in order to remove slaves. Removal will be attempted whilst unlocked.
PUSH-000506	Flow control pressure '{}' percent.	Flow control back pressure has reached this value.
PUSH-000507	Flow control off.	Back pressure has reduced, flow control is no longer being applied.
PUSH-000508	Client '{}' sent message to unknown topic '{}'.	Classic API clients can only can send messages to existing topics.
PUSH-000509	The server failed to forward a message from {} to one or more classic clients for topic {} because a failure occurred whilst creating a message.	The server failed to forward a message to one or more classic clients as it was unable to create a message. See the log for further detail.
PUSH-000510	Unable to rotate metrics file '{}'.	The internal FileMetricsWriter was unable to rotate out the specified file.
PUSH-000511	Unable to write metrics to file '{}'.	The internal FileMetricsWriter was unable to write to the specified file.
PUSH-000512	Session property {} not allowed.	An attempt has been made to define a session property with a name that is not allowed.
PUSH-000513	Failed to create load message for update to topic {}.	An error was encountered while updating topic data.
PUSH-000514	Cannot parse "server-name/sessionId" expression required for delivery to Diffusion client: {}.	The nominated routing property lacks a valid "server-name/sessionId" expression.

Id	Message	Description
PUSH-000515	Cannot place JMS session to provider '{}': {}.	An error occurred when trying to place a JMS session to the given provider.
PUSH-000516	Cannot publish message to topic {}: {}.	An error occurred while publishing a Diffusion topic update.
PUSH-000517	Cannot transform message for topic {}: {}.	Transformation between a JMS Message and a Diffusion message has failed.
PUSH-000518	Connected to JMS provider '{}'.	The JMS adapter has connected to the JMS provider with the given name.
PUSH-000519	Connection attempt to '{}' failed.	It was not possible to connect to the given JMS provider.
PUSH-000520	Exception closing JMS Connection to {}: {}.	An error occurred while the JMS connection was being closed.
PUSH-000521	Failed to deliver message to client {}: {}.	An error occurred during an attempt to deliver a message to a Diffusion client.
PUSH-000522	Failed to send message to JMS destination: {}.	An error occurred while sending a message from the JMS adapter to the JMS server.
PUSH-000523	Retrying delivery of message to {}: {}.	Following a failed delivery the delivery re-attempted once.
PUSH-000524	Returning message '{}' to origin: {}.	A message received from a client could not be delivered to a JMS destination, and is being returned to the sender.
PUSH-000525	Routing-property {} absent from message.	To relay a message to an individual Diffusion client a "server-name/sessionId" expression must be retrieved from a JMS header or property. The configured header/property is absent from this message.
PUSH-000526	Topic {} is undefined, therefore cannot receive any classic messaging.	Classic messaging mandates a topic is bound to the topic path, and in this case there is none.
PUSH-000527	Cannot establish publication to destination {}: {}.	It was not possible to build a MessageProducer for the given JMS destination.
PUSH-000528	Connected to JMS provider {} v{}, JMS {}.	Recording JMS provider and version, and JMS API version in use.
PUSH-000529	Failed to generate snapshot for topic {}.	The attempt to generate a snapshot for distribution through replication failed. See the stacktrace for more

Id	Message	Description
		information on why the snapshot could not be generated.

Related Links

Logging on page 721
The Diffusion server logs messages about its activities and any problems that occur. Log messages are written to the console (the process output, which can appear in a terminal window) and to a log file. The level of logging can be adjusted, from a moderate rate of messages under normal circumstances for system monitoring, to very detailed logging for problem diagnosis.

Introspector

An introduction to the Introspector Eclipse plugin

This section introduces and explains the features of the Diffusion Introspector GUI. This is currently a beta-product, and as such might contain bugs.

The Introspector GUI is a set of Eclipse plugins. This product depends on a minimum of Eclipse version Juno or later for Java EE Developers which can be downloaded from http://eclipse.org/downloads/.

This GUI uses features from the existing system publisher as well as the new Diffusion publisher for which a license must be obtained.

Supported platforms.

The stack of technologies upon which this software is expected to function.

Introspector is tested on version Juno or later for Java EE Developers and Java HotSpot Development Kit 8 (latest update).

Installing from update site

All Diffusion Eclipse plugins are distributed primarily through the Eclipse Update site.

The Eclipse Update site is located at the following URL: http://download.pushtechnology.com/eclipse/5.5.

A To install the software, ensure you have version Juno or later for Java EE Developers installed. Earlier versions might work, but are not supported. Pick the **Help** menu, click the **Install new software** menu item. From the dialog, choose the **Add ...** button. Complete the dialog as shown below.

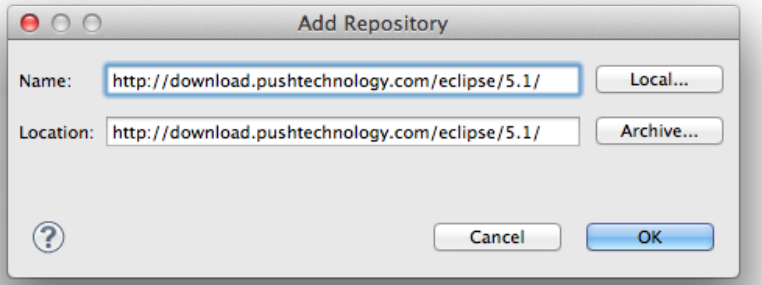


Figure 94: Adding a repository

Click on the **OK** button to go to the Available Software dialog.

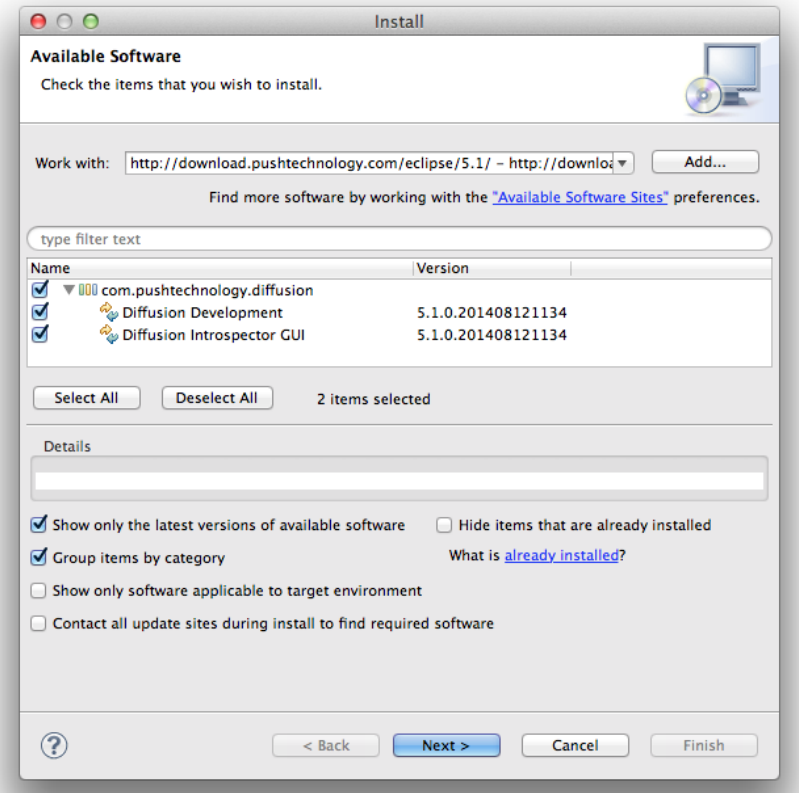


Figure 95: Install dialog

Click the **Next** button. Follow the sequential stream of dialogs. When you are given the dialog shown below, select **I accept the terms of the license agreement** if you agree to the license terms.

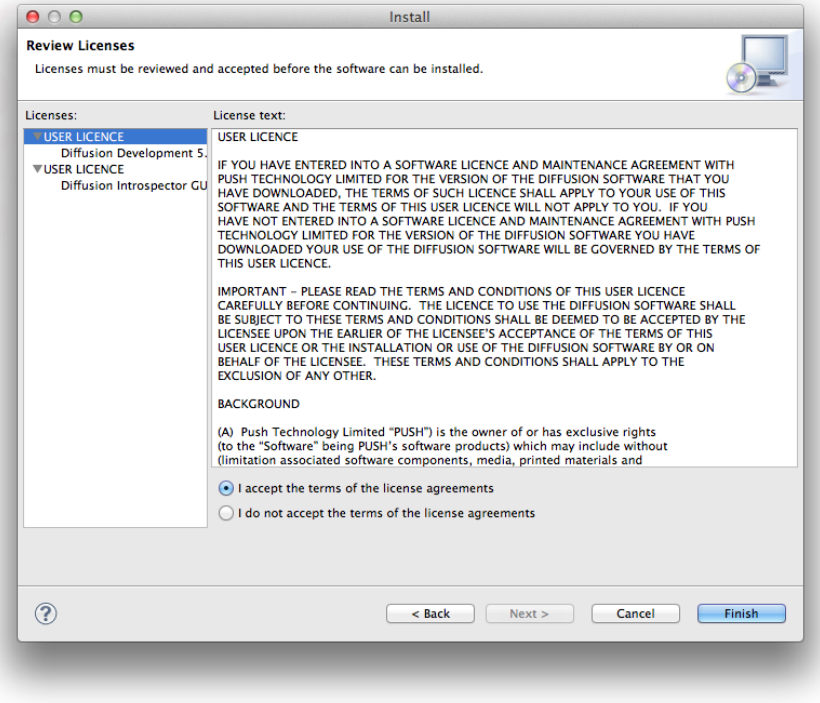

Install
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
Licenses:
License text:
USER LICENCE
Diffusion Development 5.
USER LICENCE
Diffusion Introspector GU
USER LICENCE
IF YOU HAVE ENTERED INTO A SOFTWARE LICENCE AND MAINTENANCE AGREEMENT WITH PUSH TECHNOLOGY LIMITED FOR THE VERSION OF THE DIFFUSION SOFTWARE THAT YOU HAVE DOWNLOADED, THE TERMS OF SUCH LICENCE SHALL APPLY TO YOUR USE OF THIS SOFTWARE AND THE TERMS OF THIS USER LICENCE WILL NOT APPLY TO YOU. IF YOU HAVE NOT ENTERED INTO A SOFTWARE LICENCE AND MAINTENANCE AGREEMENT WITH PUSH TECHNOLOGY LIMITED FOR THE VERSION OF THE DIFFUSION SOFTWARE YOU HAVE DOWNLOADED YOUR USE OF THE DIFFUSION SOFTWARE WILL BE GOVERNED BY THE TERMS OF THIS USER LICENCE.
IMPORTANT – PLEASE READ THE TERMS AND CONDITIONS OF THIS USER LICENCE CAREFULLY BEFORE CONTINUING. THE LICENCE TO USE THE DIFFUSION SOFTWARE SHALL BE SUBJECT TO THESE TERMS AND CONDITIONS SHALL BE DEEMED TO BE ACCEPTED BY THE LICENSEE UPON THE EARLIER OF THE LICENSEE'S ACCEPTANCE OF THE TERMS OF THIS USER LICENCE OR THE INSTALLATION OR USE OF THE DIFFUSION SOFTWARE BY OR ON BEHALF OF THE LICENSEE. THESE TERMS AND CONDITIONS SHALL APPLY TO THE EXCLUSION OF ANY OTHER.
BACKGROUND
(A) Push Technology Limited "PUSH") is the owner of or has exclusive rights (to the "Software" being PUSH's software products) which may include without (limitation associated software components, media, printed materials and
I accept the terms of the license agreements
I do not accept the terms of the license agreements
< Back
Next >
Cancel
Finish

Figure 96: Accept the license agreement

If asked to select the certificate to trust, select **Push Technology Ltd; Development; Push Technology Ltd** and click **OK**.

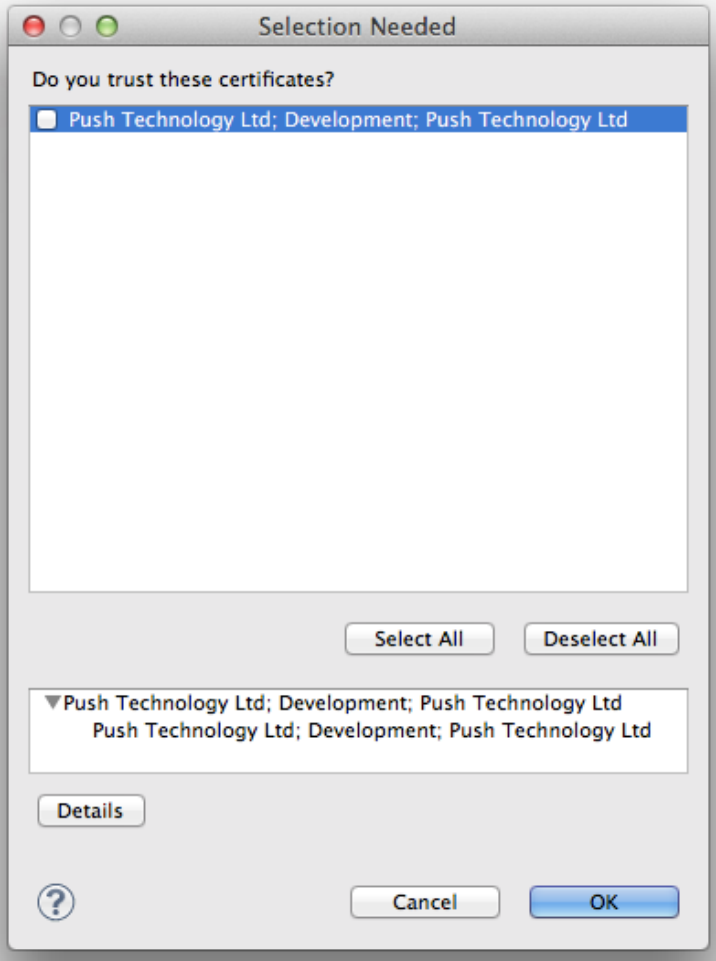


Figure 97: Click OK

When the installation process is done, click the **Restart Now** button.

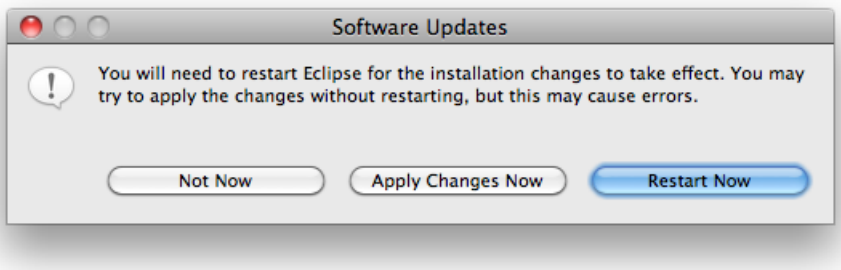

Software Updates
You will need to restart Eclipse for the installation changes to take effect. You may try to apply the changes without restarting, but this may cause errors.
Not Now
Apply Changes Now
Restart Now

Figure 98: Restarting

Installing subsequent plugin updates.

Once this process is complete users can install the latest version of the plugins by clicking on the **Help** menu and picking the **Check for updates** menu item.

Uninstalling

Eclipse maintains a registry of what plugins have been installed, and gives users a means of removing plugins from it.

Firstly, open the **About Eclipse** dialog. On OS X this can be found in the **Eclipse** menu, on every other platform – the 'Help' menu.

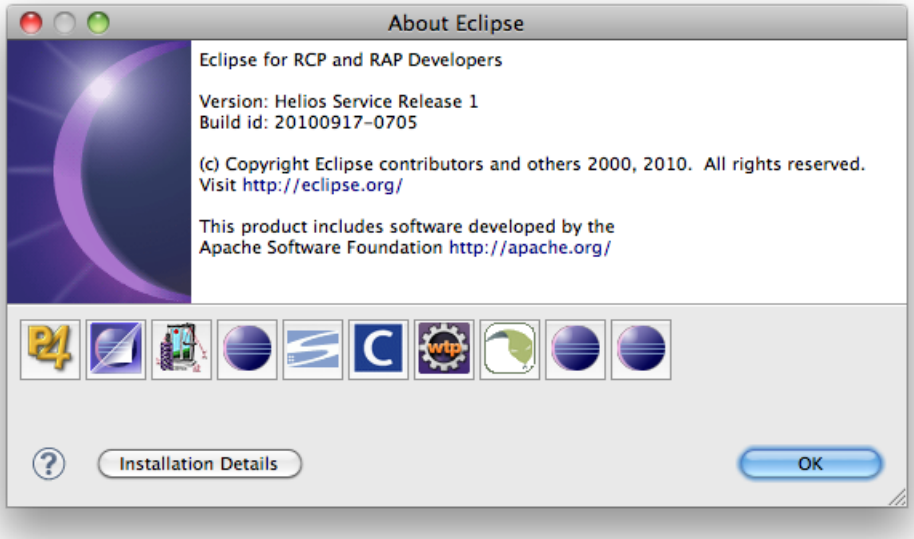

About Eclipse
Eclipse for RCP and RAP Developers
Version: Helios Service Release 1
Build id: 20100917-0705
(c) Copyright Eclipse contributors and others 2000, 2010. All rights reserved.
Visit http://eclipse.org/
This product includes software developed by the
Apache Software Foundation http://apache.org/
Installation Details
OK

Figure 99: About Eclipse dialog

Click on **Installation details**. Once the **Eclipse Installation Details** dialog appears, click on the **Installed Software** tab to review which plugins are installed.

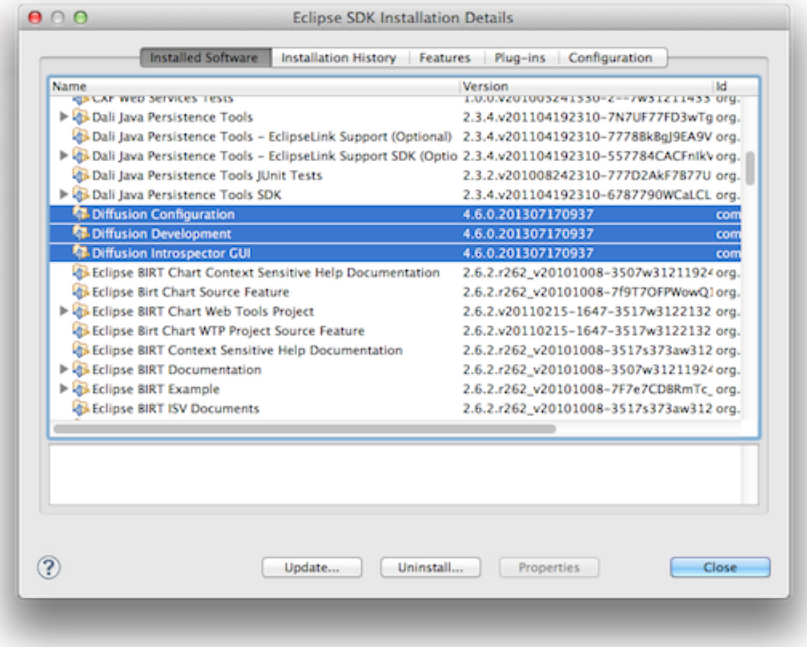


Figure 100: Installed plugins

Once a the software is found, you can uninstall it using the **Uninstall ...** button. Users might be advised to restart Eclipse once the process is complete.

Opening the Diffusion perspective

The Diffusion perspective contains a set of views that relate to Diffusion functional.

Select **Window** > **Open Perspective** > **Other ...**. Pick the **Diffusion** item from the list and click on **OK**.

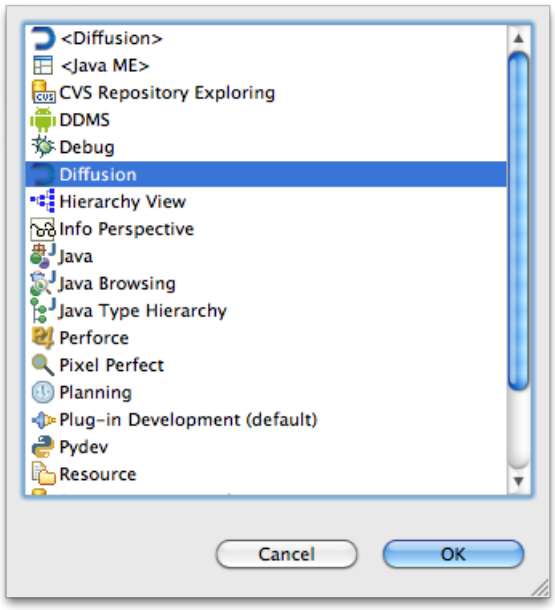


Figure 101: Perspective

Opening the Diffusion perspective presents to the user a set of views related to Diffusion functionality. The same views can be added to other perspectives (such as the Java perspective):

- Select **Window** > **Show View** > **Other ...**.
- Find the Diffusion section in the list and pick the views required.

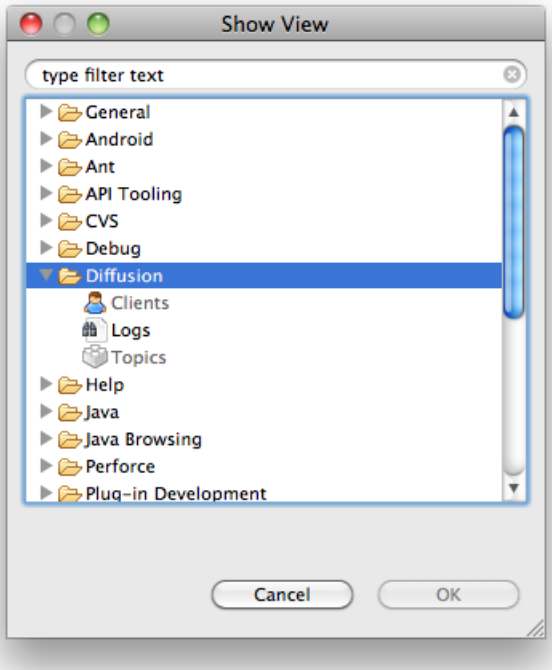


Figure 102: Views

The Diffusion perspective shows the topics and clients views on the left hand side.

Adding servers

To add server details to the plugin click the green plus button in the head of the topic view or client view.

You can click on the **Test** button to test the connection before clicking **OK**.

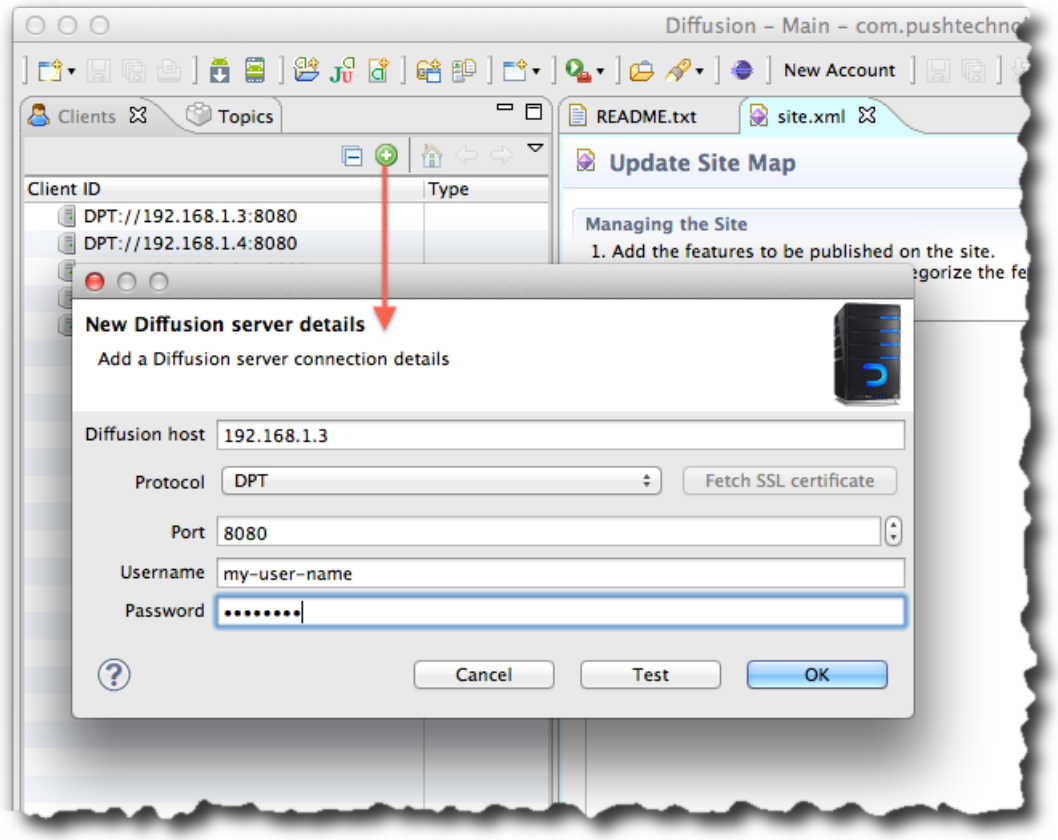


Figure 103: Add a server

If the you want to amend server details, the context menu of closed server (that is, those servers that are not currently connected) holds an **Edit** item that presents the same dialog box.

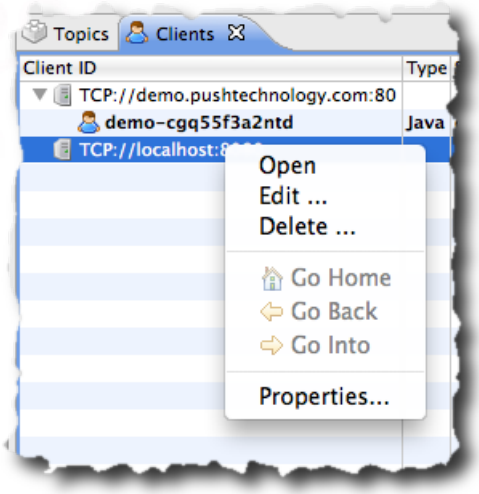


Figure 104: Edit server details

Opening servers

You can view server information through Eclipse.

In either the topics view or the clients view, right-click on a server icon and pick **Open** from the context menu, or double-click on the icon.

The set of columns shown by default show the Diffusion server name (server.name in `etc/Server.xml`) in the value column and the time that the server was started in the **Created** column.

Exploring the topics

You can navigate the tree of topics the same way as any other tree control.

The GUI populates the tree on-demand and asynchronously, so a branch of the tree might be populated some short time after it has been opened. Each topic shows its name, the publisher, the type of any topic-data associated, the date-stamp the topic was created, the date-stamp the topic was last updated and the value of that topic. The last two are populated only if the topic is fetched or subscribed.

The topic tree is live and as new topics are created and old topics deleted so the tree is updated.

Getting topic values

Users can retrieve the current state of the topic by picking **Fetch** from the context menu of a topic. Picking **Fetch recursively** also fetches the values of all topics below this one.

Users can also subscribe to a topic by picking **Subscribe** from the context-menu of a topic. **Subscribe recursively** also subscribes to values of all topics below this one. When the value of a topic changes the **Value** and **Updated** columns flash blue for a second.

To drill into a part of the topic tree they can pick **Go Into** from the topic context menu. The view shows the name of the top-level topic in the view header.

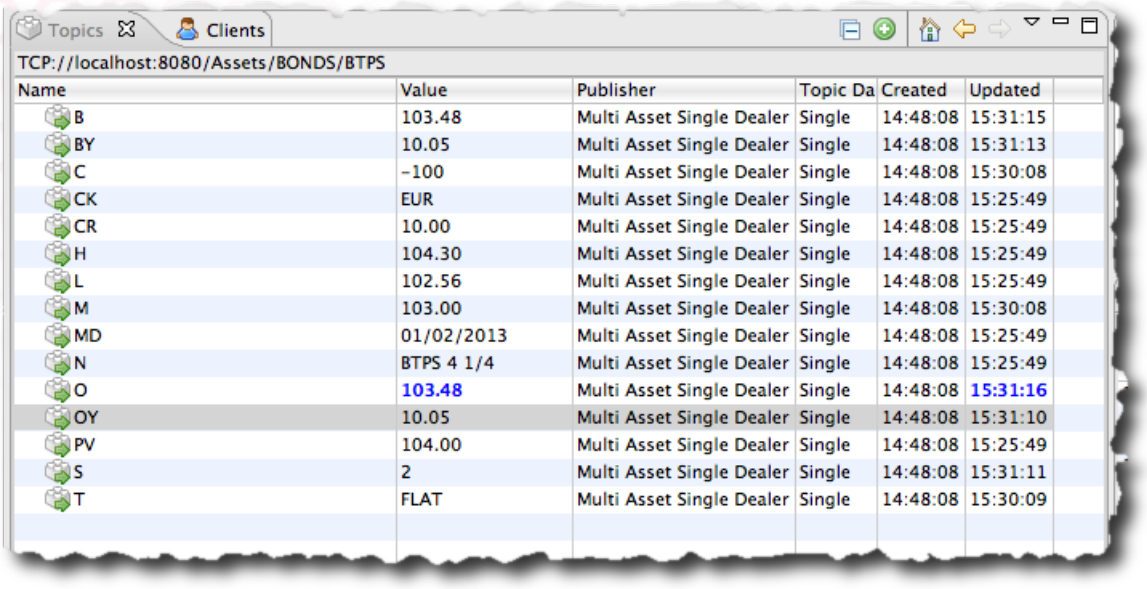


Figure 105: View topic values

Click the back button to return to the previous position in the topic tree.

Configuring columns

Columns can be reordered with drag and drop. Column positions are stored between uses. The context menu of the headers lets users toggle individual columns on and off.

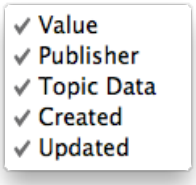


Figure 106: Re-order columns

Ping servers

To establish the response time of a remote server pick **Ping** from the context menu of a connected server. The results in milliseconds is shown next to the server name in the **Name** column.

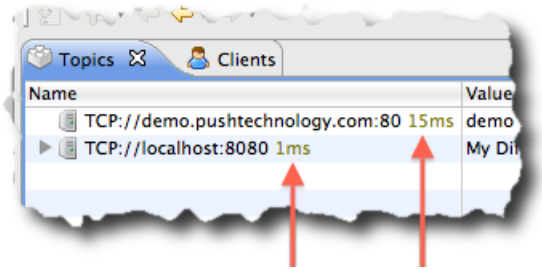


Figure 107: Ping a server

Count topics

Pick **Count topics** from the context-menu of a topic for the server to count the number of topics underneath this one.

The result is shown to the right of the topic name

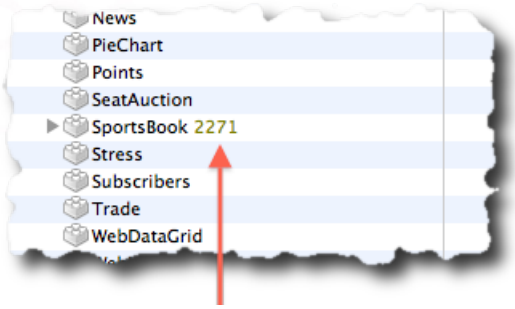


Figure 108: Topic count

Using the clients view

The clients view is a live view of the clients connected to the set of Diffusion servers.

As the Diffusion GUI is also a Diffusion client, it shows up in this list and is shown in bold. A newly connected client is bright blue for 1s, while a disconnecting client is drawn pale gray for 1s before it vanishes.

The properties of a client listed in the following table.

Table 90: Client properties in the Eclipse client view

Type	Flash, Silverlight, WebSockets, iOS, or Android, and so on.
Server	the name of the server this client is connected to
Start time	the time-stamp the client connected to its server
Resolved name	the result of a WHOIS query on the IP address of the remote client
Description	more details from the WHOIS query
Locale	The nation or territory from which this client is connecting
Ping	The time taken to ping the remote client
Alarm	Not currently in use
IP address	The IP address of the remote client
Client reference	The reference/description granted to a client

Ping

Using this feature the user can ping individual clients, the results of which show up in the **Ping** column

All Diffusion clients respond to ping requests from the server.

Further details of a client can be gained from the properties view. Pick **Properties** from the context menu of a client.

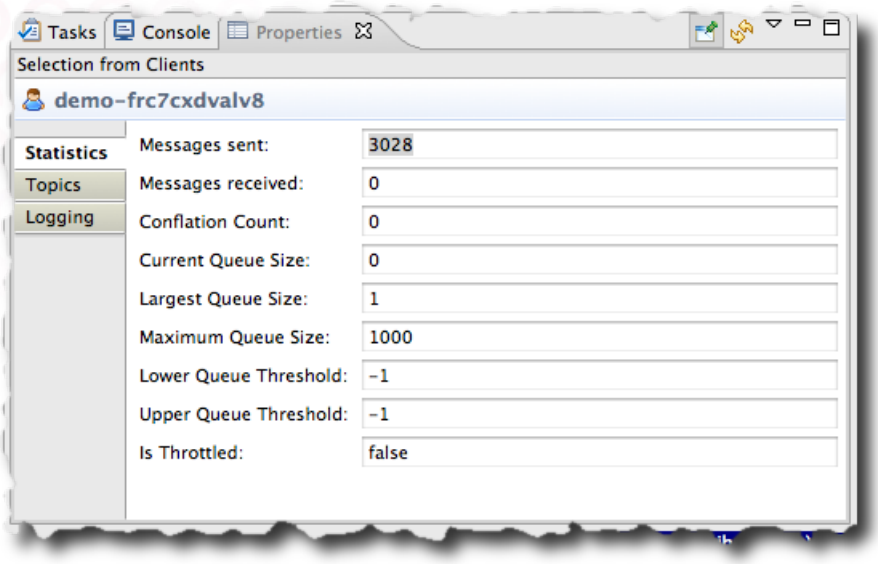


Figure 109: Ping clients

The **Properties** view is part of the Eclipse framework and used in many other non-Diffusion circumstances. Consequently, if the user clicks a new object in another view the **Properties** view might show the properties of that object instead.

Statistics

This shows the number of messages sent, received, the number of messages conflated, the current queue size, the highest queue size, the maximum queue size allowable and throttling thresholds.

Unlike previous views this data is not live, and the refresh button in the view header must be clicked to refresh this view.

Topics

This section shows the topics to which this client is currently subscribed along with any related topic reference. This data is also not live, and must be refreshed manually.

Logging

Access to remote client logs is currently an experimental and optional feature. Those clients that do support it subscribe to the Diffusion/ClientLogs topic.

Using this section of the view, you can retrieve log entries from remote JavaScript clients that are normally sent to the JavaScript console alone.

The JavaScript environment inside a modern browser is memory constrained. This facet of the GUI lets the user set the level at which a log entry is stored and the size of the buffer in which entries are stored before being removed.

Server logs

Diffusion server logs are published through the Diffusion/Logs topic as well as written to log files. The Logs view subscribes to Diffusion to display the log entries as they happen.

Right-click on any open Diffusion server item in the topics or clients view and pick **Logs ...** from the context menu.

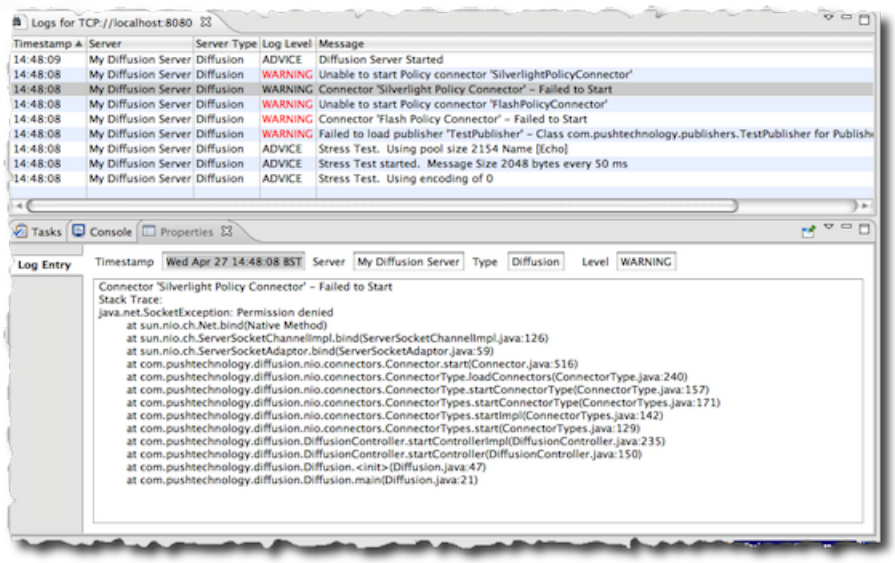

Logs for TCP://localhost:8080
Timestamp
Server
Server Type
Log Level
Message
14:48:09 My Diffusion Server Diffusion ADVICE Diffusion Server Started
14:48:08 My Diffusion Server Diffusion WARNING Unable to start Policy connector 'SilverlightPolicyConnector'
14:48:08 My Diffusion Server Diffusion WARNING Connector 'Silverlight Policy Connector' – Failed to Start
14:48:08 My Diffusion Server Diffusion WARNING Unable to start Policy connector 'FlashPolicyConnector'
14:48:08 My Diffusion Server Diffusion WARNING Connector 'Flash Policy Connector' – Failed to Start
14:48:08 My Diffusion Server Diffusion WARNING Failed to load publisher 'TestPublisher' – Class com.pushtechnology.publishers.TestPublisher for Publish
14:48:08 My Diffusion Server Diffusion ADVICE Stress Test. Using pool size 2154 Name [Echo]
14:48:08 My Diffusion Server Diffusion ADVICE Stress Test started. Message Size 2048 bytes every 50 ms
14:48:08 My Diffusion Server Diffusion ADVICE Stress Test. Using encoding of 0
Tasks
Console
Properties
Log Entry
Timestamp
Wed Apr 27 14:48:08 BST
Server
My Diffusion Server
Type
Diffusion
Level
WARNING
Connector 'Silverlight Policy Connector' – Failed to Start
Stack Trace:
java.net.SocketException: Permission denied
at sun.nio.ch.Net.bind(Native Method)
at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:126)
at sun.nio.ch.ServerSocketAdaptor.bind(ServerSocketAdaptor.java:59)
at com.pushtechnology.diffusion.nio.connectors.Connector.start(Connector.java:516)
at com.pushtechnology.diffusion.nio.connectors.ConnectorType.loadConnectors(ConnectorType.java:240)
at com.pushtechnology.diffusion.nio.connectors.ConnectorType.startConnectorType(ConnectorType.java:157)
at com.pushtechnology.diffusion.nio.connectors.ConnectorTypes.startConnectorType(ConnectorTypes.java:171)
at com.pushtechnology.diffusion.nio.connectors.ConnectorTypes.startImpl(ConnectorTypes.java:142)
at com.pushtechnology.diffusion.nio.connectors.ConnectorTypes.start(ConnectorTypes.java:129)
at com.pushtechnology.diffusion.DiffusionController.startControllerImpl(DiffusionController.java:235)
at com.pushtechnology.diffusion.DiffusionController.startController(DiffusionController.java:150)
at com.pushtechnology.diffusion.Diffusion.<init>(Diffusion.java:47)
at com.pushtechnology.diffusion.Diffusion.main(Diffusion.java:21)

Figure 110: Server log entries

By clicking on the **Timestamp** table header, users can order the log entries by time-stamp. The pull down menu on the right hand side of the view header lets the user filter log entries by the entry log-level.

Note: This does not affect the logging from the server as this is a client-side filter only.

You can get the properties of server log entry from the context menu of an entry. Shown above are the properties of a multi-line log-entry.

Property obfuscator

This dialog is part of the Diffusion perspective and can be used to hide sensitive Diffusion configuration file entries, such as passwords and JMS login credentials.

Select or enter into the dialog the text that you want to obfuscate. All obfuscated entries start with `OB:`. The **Copy** button copies the obfuscated text to the clipboard, ready for pasting back into the configuration file.

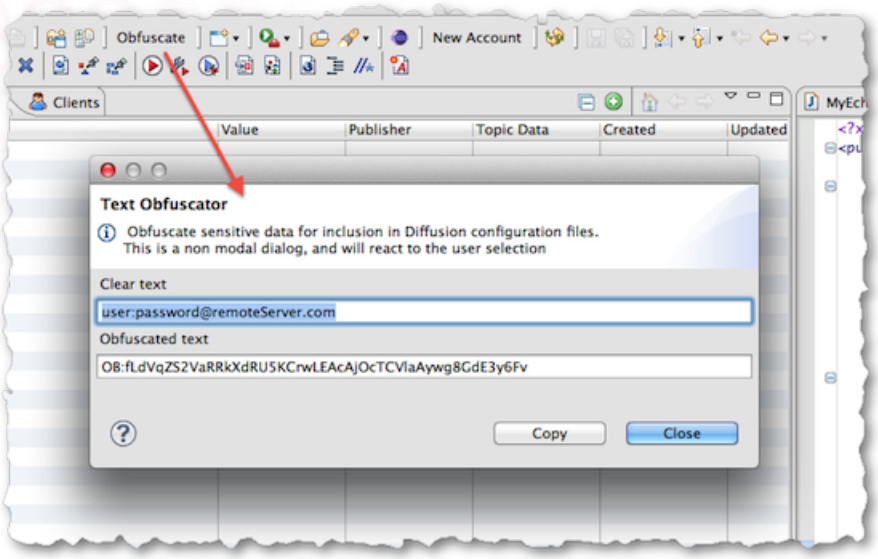


Figure 111: Property Obfuscator dialog

The dialog is modeless, and reacts to the users selection changes you do not need to dismiss the dialog if you have more than one value to obfuscate.

Demos

Diffusion comes with demo applications that demonstrate certain features of Diffusion.

If you have chosen to include demos as part of the installation process, the demos are included as DAR files in the `demos` directory of your Diffusion installation and as source in the `demos/src` directory of your Diffusion installation. You can build the demos from source using *mvndar*.

Demos

The demos in the **Demos** section of the default Diffusion installation web page demonstrate various applications of Diffusion. They also contain a connection widget, allowing you to modify the transports used as well as inspect messages incoming and outgoing from the client.

To access the Diffusion web page and demos, start the web server and type in the following URL into a browser: http://localhost:8080.

Table 91: Demos provided with the Diffusion server

Tech Magnets demo	This demo provides an example of shared state handled by Diffusion. Tiles are created on the server in response to client actions. When a tile is moved by the client, a message is sent to the Diffusion server, the state is modified and an exclusive message is sent to all other subscribers. Open this demo in multiple browsers to see the real-time DOM updates.

Drawing Board demo	Multiple users draw on a virtual blackboard with colored chalk. All clients are updated in real-time with strokes from the other clients. Open this demo in multiple browsers to see the responsiveness of client interactions. Requires Canvas support in the browser.
Dogfight (game) demo	Aerial combat game for up to 8 players. Use the arrow keys to control your plane and the space key to fire. Uses HTML 5 features.
Asset Trader	View and trade on our cross-platform FX trading demo where price feeds, instruments, news, and user interactions are all distributed in real-time.

Building the demos using mvndar

You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

Building a DAR file using mvndar (recommended)

The mvndar plugin for Maven is developed and maintained on GitHub: https://github.com/pushtechnology/mvndar.

To use mvndar, reference it in the `plugins` section of your `pom.xml` file.

```
<plugins>
      ...
      <plugin>
        <groupId>com.pushtechnology.tools</groupId>
        <artifactId>dar-maven-plugin</artifactId>
        <version>1.2</version>
        <extensions>true</extensions>
      </plugin>
      ...
    </plugins>
```

The plugin runs during the project's package phase and creates the DAR file in the target directory.

Building the demos

The `pom.xml` file included in the `demos/src` directory of your Diffusion already contains the required reference to mvndar.

To build the demos using mvndar, complete the following steps:

1. From the command line, go to the *`diffusion_installation`*`/demos/src` directory.
2. Set the `DIFFUSION_HOME` environment variable to the absolute path of your Diffusion installation.
   - On Linux or OS X, type `export DIFFUSION_HOME=`*`/diffusion_installation`*
   - On Windows, type `set DIFFUSION_HOME=`*`C:/diffusion_installation`*
3. Run the `mvn clean install` command.

Related Links

Using Maven to build Java Diffusion applications on page 786
Apache Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

Building publishers and other server application code with Maven on page 788
The Diffusion API for publishers and other server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

Tools

If the tools were installed during the installation process, there are some tools that can help with the monitoring of Diffusion plus some handy utilities.

There are additional tools and utilities that are available in public repositories, such as GitHub and Maven.

Tools for Amazon Elastic Compute Cloud (EC2)

This is a description of a number of tools and adaptations which are provided for use when using Diffusion with Amazon EC2™.

Diffusion includes in `tools/ec2/` a number of files useful when deploying Diffusion to an Ubuntu image on an EC2 virtual machine.

diffusion.conf

Ubuntu makes use of the Upstart daemon as a replacement for `init`. Copied to `/etc/init/diffusion.conf` this file contains configuration for Upstart to begin Diffusion at boot-time in a background process as a unprivileged user. It establishes iptables rules to route traffic from privileged ports to Diffusion.

Users can stop and start Diffusion using Upstart commands, for example, to start the server

```
service diffusion start
```

To stop Diffusion

```
service diffusion stop
```

To check the status of Diffusion

```
service diffusion status
```

In the event that something goes amiss Upstart writes a log file to `/var/log/upstart/diffusion.log`

etc/Connectors.xml

Except for two port number changes this is an otherwise regular copy of `etc/Connectors.xml` normally found in a Diffusion installation. This edition however binds the Flash policy connector and the Silverlight policy connector to port numbers greater that 1024, making it possible to run Diffusion as an unprivileged process. Use this file in conjunction with the iptables rules established in `diffusion.conf`

`ec.xml`

An illustrative Apache Ant script that can be used to start, stop, and get status from a Diffusion server running on an Amazon EC2 Linux host. It also demonstrates an inelegant means of deploying and undeploying a DAR file to/from the remote server, by copying the file to the remote server, then moving it into the deploy directory.

Table 92: Targets

Property	Purpose
start	Runs `sudo service diffusion start` on remote host using SSH
stop	Runs `sudo service diffusion stop` on remote host using SSH
status	Runs `sudo service diffusion status` on remote host using SSH
deploy	Uploads local DAR file to staging dir, then moves it into Diffusion deploy directory
undeploy	Removes DAR file from deploy directory, signaling Diffusion to undeploy related publishers (where possible)

The script is driven by named properties:

Table 93: Properties for targets start, stop and status

Property	Purpose
remote.host	EC2 host running sshd
remote.username	Authentication username, default of `ubuntu`
remote.keyfile	PEM encoded key use during authentication

Table 94: Additional properties for targets deploy and undeploy

Property	Purpose
dar.file	Name of a DAR file to deploy or undeploy
remote.diffusion.dir	Root directory of the remote Diffusion installation

Example deployment:

```
ant -Dremote.host=54.235.65.36 \
 -Dremote.keyfile=$HOME/.ssh/ec2-push1.pem \
 -Ddar.file=$HOME/Applications/Diffusion5.5.3/demos/dogfight.dar
 \
 -Dremote.diffusion.dir=/home/ubuntu/Diffusion/Diffusion5.5.3 \
 deploy
```

The script uses no proprietary Diffusion code and is open to extension during development of a solution. Both the `sshexec` and `scp` tasks depend on the jsch library which might have to be downloaded.

Tools for Joyent

This is a description of adaptations which are provided for use when using Diffusion within the Joyent Cloud.

Diffusion includes in `tools/joyent/` files useful when deploying Diffusion to an SmartOS virtual machine in the Joyent cloud.

`diffusion.xml`

SmartOS makes use of SMF to start services at boot-time. Once tailored for the host and installed into SMF this file contains configuration for SMF to begin Diffusion at boot-time in a background process as an unprivileged user called `push`. This file does not create or assert the existence of a `push` user.

Users can stop and start Diffusion using SMF commands. For example, to start the server

```
sudo svcadm enable diffusion
```

To stop the server

```
sudo svcadm disable diffusion
```

Using Maven to build Java Diffusion applications

Apache Maven is a popular Java build tool and is well supported by Java IDEs. You can use Apache Maven to build your Diffusion applications.

The Push Technology public Maven repository

Push Technology publishes Diffusion components and related artifacts to a public Maven repository at the following location: http://download.pushtechnology.com/maven.

The published artifacts include the following:

Table 95: Artifacts

Artifact	Maven coordinates	Description
Diffusion Unified API	`com.pushtechnology.diffusion:diffusion-api:jar:5.5.3`	The Diffusion Unified API interfaces only. Use this artifact for compilation only. The JAR includes the source and Javadoc attachments.
Diffusion Clients	`com.pushtechnology.diffusion:diffusion-client:jar:5.5.3`	The Diffusion client library. The runtime library includes both the Unified API and the Classic API.
mvndar	`com.pushtechnology.tools:dar-maven-plugin:maven-plugin:1.2`	A Maven plugin for building DAR files.

To use the Push Technology public Maven repository, add the following repository description to your `pom.xml` file:

```
<repositories>
```

```
    <repository>
        <id>push-repository</id>
        <url>https://download.pushtechnology.com/maven/</url>
    </repository>
</repositories>
```

Related Links

Client API on page 500
The client API provides the ability to connect to a Diffusion server as an external client from within any Java application.

Building a DAR file on page 665
You can use the Maven plugin *mvndar* to build and deploy your publisher DAR file. Alternatively, you can use an Ant build file to perform the same function.

Building the demos using mvndar on page 783
You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

Building client applications with Maven

You can build and run Diffusion Java client applications without installing the Diffusion product. The Diffusion client JAR is all you need.

The following `pom.xml` shows how to declare the appropriate dependencies:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>myclient</artifactId>
    <version>1.0-SNAPSHOT</version>

    <repositories>
        <repository>
            <id>push-repository</id>
            <url>https://download.pushtechnology.com/maven/</
url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>com.pushtechnology.diffusion</groupId>
            <artifactId>diffusion-client</artifactId>
            <version>5.5.3</version>
        </dependency>
    </dependencies>
</project>
```

Building publishers and other server application code with Maven

The Diffusion API for publishers and other server application code is not available in the Push Technology public Maven repository. To build server components, you must install the product locally and depend on `diffusion.jar` using a Maven system scope.

The following `pom.xml` shows to declare the dependency on `diffusion.jar`. To use it, you must set the `DIFFUSION_HOME` environment variable to the absolute file path of your Diffusion installation.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.examplecorp</groupId>
    <artifactId>mypublisher</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencyManagement>
        <dependency>
            <groupId>com.pushtechnology.diffusion</groupId>
            <artifactId>diffusion</artifactId>
            <version>local-installation</version>
        </dependency>
    </dependencyManagement>

    <profiles>
        <profile>
            <activation>
                <property>
                    <name>env.DIFFUSION_HOME</name>
                </property>
            </activation>

            <dependencyManagement>
                <dependencies>
                    <dependency>
                        <groupId>com.pushtechnology.diffusion</
groupId>
                        <artifactId>diffusion</artifactId>
                        <version>local-installation</version>
                        <scope>system</scope>
                        <systemPath>${DIFFUSION_HOME}/lib/
diffusion.jar</systemPath>
                    </dependency>
                </dependencies>
            </dependencyManagement>
        </profile>
    </profiles>
</project>
```

mvndar

mvndar is a Maven plugin for creating DAR files. More information about mvndar is available at the following locations:

- http://www.pushtechnology.com/2013/08/13/mvndar-maven-plugin/
- http://pushtechnology.github.io/mvndar/index.html

Example: Using Maven to build the demo applications

If you selected the demo applications when you installed Diffusion, the source files and an example Maven `pom.xml` can be found in the directories beneath the `demos/src` directory. The example uses mvndar, and depends on `diffusion.jar` using system scope.

For more information, see Building the demos using mvndar on page 783.

Related Links

Building a DAR file on page 665
You can use the Maven plugin *mvndar* to build and deploy your publisher DAR file. Alternatively, you can use an Ant build file to perform the same function.

Building the demos using mvndar on page 783
You can use the Maven plugin *mvndar* with the provided `pom.xml` file to build the demos.

Appendix

Appendices

In this section:

- Document conventions
- Glossary
- Trademarks
- Copyright Notices

Appendix A

Document conventions

This user manual uses certain typographic conventions to distinguish between different types of information.

The following table describes how different types of information are represented typographically.

Table 96: Typographic conventions used in this manual

Convention	Usage
`Monospace`	Indicates the following items: • Source code • Class or method names • Command names • File paths • Information input by the user
Bold	Indicates the following items: • Interface element titles (buttons, menu items, field names) • Window or panel titles
Italic	Indicates the following items: • New terms – when appearing in the text • Variable values – when appearing in code or syntax examples
Greater than sign (>)	Indicates a menu item or sequence of menu items. For example, “Choose **File** > **Save**” means choose the **Save** item from the **File** menu.
Highlighting	Indicates an example value in descriptive text.

Appendix B

Glossary

The glossary contains key terms associated with Diffusion and their definitions.

In this section:

- A
- C
- D
- E
- F
- G
- H
- I
- J
- L
- M
- N
- P
- Q
- R
- S
- T
- U
- V
- W
- X

A

acknowledgment

A message sent by the recipient of a message or update to inform the sender of that message or update that it was received.Not all messages or updates are acknowledged. A message or update must specify that acknowledgment is required to receive an acknowledgment.

ack

API

Application Programming Interface
A set of contracts that you can program against to interact with Diffusion.

The following APIs are available:

- Publisher API
- Unified API
- Classic API

API

API

API

APR

Apache Portable Runtime

Apache Portable Runtime (APR)

APR

APR

ASCII

American Standard Code for Information Interchange

A character-encoding scheme that encodes 128 specified characters into 7-bit binary integers.

ASCII

ASCII

ASCII

C

callback

An object, specific to a single call, that is used to respond to a request.

certify

Push Technology certifies specific hardware and software version for use with Diffusion.
Certified versions have been fully functional tested and performance tested with the Diffusion

server. In addition, Push Technology supports some hardware and software that has not been certified.

client

An entity that connects to the Diffusion server and subscribes to topics.

Typically a client is a user-written application communicating with the Diffusion server through a client API or the Diffusion protocol. A publisher can be a client of another publisher in a distributed environment.

client library

A library that is included in a client application to enable interaction with the Diffusion server.

conflation

The merging or replacing of a queued update with a newer update to reduce network traffic. Conflation removes outdated information from the queue of content to be sent and either replaces the outdated information with the conflated information or appends the conflated information to the end of the queue.

connector

A configured point of connection to a server.

There can be one or more connectors (each listening on a different port). Each connector can accept single or multiple types of connection.

consume

When a message or update is received by a topic listener and that listener chooses not to pass on the message or update to subsequent topic listeners. Topic streams cannot choose to consume messages or updates they receive.

CORS

Cross-Origin Resource Sharing

cross-origin resource sharing (CORS)

CORS

CORS

CPU

Central Processing Unit

CPU

CPU

CPU

credentials

A piece of information that is used to authenticate a principal.

CSR

Certificate Signing Request

certificate signing request (CSR)

CSR

CSR

CSS

Cascading Style Sheet

CSS

CSS

CSS

D

DAR file

A Diffusion archive file. This file contains a publisher and can be deployed on the Diffusion server.

DAR

DAR

DAR

delimiter

A byte value that is used as a separator in messages or updates.

Depending on the type of message or update, it can contain field delimiters or record delimiters.

delta

Data that is sent to a client subscribed to a topic. The delta contains only information that has been updated on the topic since the last data was sent to the client. Topics that contain only a single item of data cannot use delta messages.

Diffusion

The Diffusion product comprising the Diffusion server and client libraries.

dirty

The state of a page in a paged topic at the client if its content has been changed on the Diffusion server, but not updated on the client.

DLL

Dynamic Link Library

DLL

DLL

DLL

DMZ

De-militarized Zone

de-militarized zone (DMZ)

DMZ

DMZ

DPT

Diffusion Protocol for TCP A Diffusion proprietary streaming protocol that clients can use to communicate with the Diffusion server.

DPT

Diffusion Protocol for TCP

DPT

E

EULA

End User License Agreement

EULA

EULA

EULA

F

feature

An API module that contains a conceptual set of facilities.

fetch

A request from a client for the current state of all data on a topic. A client can fetch a topic's state without being subscribed to the topic. This request-response mechanism of getting data from a topic is separate from the pub-sub mechanism.

field

A section of content that contains data of a specific type. Fields are nested inside records. A record can contain one or many fields.

functional topic

A topic to which data cannot be published. These topics provide other capabilities to subscribing clients, for example, notifications.

The following types of topic are functional topics:

- Routing
- Child list
- Service
- Topic notify

G

GBE

Gigabit Ethernet

GBE

GBE

GBE

global-scoped permission

Permissions at global scope apply to actions on the Diffusion server.

GUI

Graphical User Interface

GUI

GUI

GUI

H

handler

A handler is an object responsible for responding to one or more instances of a single type of request.

HDD

Hard Disk Drive

HDD

HDD

HDD

HTML

Hypertext Markup Language

HTML

HTML

HTML

HTTP

Hypertext Transfer Protocol

HTTP

HTTP

HTTP

I

IDE

Integrated Development Environment

integrated development environment (IDE)

IDE

IDE

ISAPI

Internet Server Application Programming Interface

ISAPI

initial topic load

The data sent to a client when it first subscribes to a topic. This data contains a snapshot of the current state of the topic.

initial topic load (ITL)

ITL

ITL

J

JAR

Java Archive

JAR

JAR

JAR

JDK

Java Development Kit

Java Development Kit (JDK)

JDK

JDK

JMS

Java Message Service

Java Message Service (JMS)

JMS

JMS

JMX

Java Management Extensions

Java Management Extensions (JMX)

JMX

JMX

JRE

Java Runtime Environment

Java Runtime Environment (JRE)

JRE

JRE

JVM

Java Virtual Machine

Java Virtual Machine (JVM)

JVM

JVM

L

LDAP

Lightweight Directory Access Protocol

Lightweight Directory Access Protocol (LDAP)

LDAP

LDAP

line

An entry in a page of a paged topic.The number of lines on a page is specified by the client that views the paged topic.

listener

In the Unified API, a listener is an object that is always called when a particular event occurs.

M

message

A message is a series of bytes of information formatted according to the Diffusion protocol which can be sent between components utilizing Diffusion.

message queue

A queue of messages. Each client connection to Diffusion has such a queue on the Diffusion server upon which messages are put for sending to the client.

queue

metadata

Data about data. In Diffusion metadata is used to define the structure of messages.

message metadata

multicast

To send data to several recipients at the same time.

The datagrid uses multicasting to locate other datagrid nodes.

N

NAT

Network Address Translation

network address translation (NAT)

NAT

NAT

NIC

Network Interface Controller

NIC

NIC

NIC

NIO

New Input-Output

NIO

NIO

Non-blocking Input/Output

NIO

notification

...

P

paged topic

A topic whose data is formatted in lines and viewed as pages containing one or more lines.

The following types of topic are paged topics:

- Paged string
- Paged record

PNG

Portable Network Graphics

PNG

PNG

PNG

permission

A permission represents the right to perform an action on the Diffusion server or on data hosted by the Diffusion server. Permissions can be global- or topic-scoped.

PID

Process ID

PID

PID

PID

ping

A query sent by a publisher, client or by the Diffusion server to a connected component to check that the connection exists and the latency of the connection.
The following types of ping are available:

server ping
: A client pings the Diffusion server.

client ping
: A publisher pings a specific client.

system ping
: Diffusion pings all clients at a regular interval.

PDF

Portable Document Format

PDF

PDF

PDF

primary server

In a fan-out solution, the server from which updates are fanned out to replica servers.

In previous releases, this server was called the master server. This terminology in no longer used.

master server

principal

An identity that can be authenticated by the Diffusion server or by a client.

A principal can be a user or client. After a principal has been authenticated, it can be assigned roles that enable it to access actions or resources.

protocol

A protocol defines the exact format of data passed between the Diffusion server and a client.

publisher

The component which publishes messages relating to one or more topics.

A server can host one or more publishers. Messages sent by clients on particular topics are routed to the Publisher that owns the topic. Publisher functionality is provided by users by writing a Java publisher class.

publishing topic

A topic where data is published and from which the data is distributed to subscribing clients.

The following types of topic are publishing topics:

- Single value

- Record
- Stateless
- Delegated (Deprecated)
- Protocol buffer
- Custom

Q

message queue

A queue of messages. Each client connection to Diffusion has such a queue on the Diffusion server upon which messages are put for sending to the client.

queue

R

RAID

Redundant Array of Independent Disks

RAID

RAID

RAID

RAM

Random Access Memory

RAM

RAM

RAM

record

A section of content that acts as a container for a set of fields. Inside the content of a message or update you can have zero, one, or many records. A record can contain one or many fields.

regular expression

A string that uses special characters to describe a search pattern.

Diffusion uses Java-style regular expressions.

regex

replica server

In a fan-out solution, a server to which updates are fanned out from the primary server.

In previous releases, this server was called the slave server. This terminology in no longer used.

slave server

RMI

Remote Method Invocation

remote method invocation (RMI)

RMI

RMI

role

A role is a named set of permissions and other roles. Principals and sessions can both be assigned roles.

role hierarchy

Roles are hierarchical. A role can include other roles and, by doing so, have the permissions assigned to the included roles. A role cannot include itself, either directly or indirectly – through a number of included roles.

RPM

Redhat Package Manager

Redhat Package Manager (RPM)

RPM

RPM

S

SAS

Serial Attached SCSI

SAS

SAS

SAS

SDK

Software Development Kit

software development kit (SDK)

SDK

SDK

server

The component that hosts topics and publishers. A server broadcasts topic updates to all subscribed clients.

Clients can connect to servers through the Unified API or Classic API.

Diffusion server

session

An ongoing dialog between a client and the Diffusion server.

Typically, a session represents a single client connection to a single server. However, in the event of connection failure the session can automatically reconnect to the same server or even failover to another server and still retain its context.

SLF4J

Simple Logging Facade for Java

SLF4J

SLF4J

SLF4J

SSH

Secure Shell

SSH

SSH

SSH

SSL

Secure Sockets Layer

Secure Sockets Layer (SSL)

SSL

SSL

state

The latest published values of all data items on the topic. The state of a topic is stored on the Diffusion server.

structural conflation

A form of conflation that enables you to define the operations performed on outdated content. You can merge, aggregate, reverse or combine the effects of multiple changes into a single consistent and current notification to the client.

stream

In the Unified API, a stream is a sequence of responses to a single request.

subscribe

A client registers interest in a topic such that the client receives messages sent to that topic.

support

Push Technology supports a number of hardware and software versions, these versions have not necessarily been tested. Those hardware and software versions that we have tested are listed as 'certified'.

T

TCP

Transmission Control Protocol

TCP

TCP

TCP

throttling

Limiting the volume of messages that the Diffusion server transmits to a client within a specified period of time.

Throttling can be used to limit bandwidth usage or to prevent more messages being sent to a client than the client can handle.

TLS

Transport Layer Security

Transport Layer Security (TLS)

TLS

TLS

topic

A logical channel through which messages are distributed.

Topics provide a logical link between publishers and subscribers. Clients or publishers publish messages to topics. Clients subscribe to topics to receive messages published to that topic.

topic path

A string representation of a location in the topic tree.

A topic path consists of parts separated by a slash character (/).

Topic paths describe a location where a topic can be bound and used for pub-sub distribution of data.

Topic paths can also be used for bi-directional messaging: a client can send a message to a topic path and the Diffusion server routes the message to the topic's publisher or publishers.

topic name

hierarchic topic name

full topic name

topic path prefix

The root part of a topic selector.

A concrete topic path to the most specific part of the topic tree that contains all topics that the selector can specify. For example, for the topic selector `?foo/bar/baz/.*/bing`, the topic path prefix is `foo/bar/baz`.

path prefix

topic selector

An object that retrieves one or more topics based on their topic paths.

A topic selector uses a pattern expression, which can include one or more regular expressions, to match to the path of one of more topics.

selector

topic-scoped permission

Permissions at topic scope apply to actions on a topic.

Topic-scoped permissions are defined against topic branches. The permissions that apply to a topic are the set of permissions defined at the most specific branch of the topic tree.

topic tree

The organization structure of topics on the Diffusion server.

A topic can have subtopics and can itself be a subtopic of another topic. All topics created on the Diffusion server by a publisher or client are in the topic tree.

topic hierarchy

transport

An implementation of a network protocol. The mechanism by which clients communicate with the Diffusion server.

U

update

Data published to a topic by a client or publisher that is applied to the topic to change the topic state. The updated data is then pushed out to all subscribing clients.

URL

Uniform Resource Locator

URL

URL

URL

UTF-8

Universal Character Set Transformation Format 8-bit

A character encoding capable of encoding all possible characters in Unicode.

UTF-8

UTF-8

UTF-8

V

VCPU

Virtual Central Processing Unit

VCPU

VCPU

VCPU

W

WAR

Web Application Archive

WAR

WAR

WAR

X

XHR

XmlHttpRequest

XHR

XHR

XHR

XML

Extensible Markup Language

XML

XML

XML

XSD

XML Schema Definition

XSD

XSD

XSD

Appendix C

Trademarks

The following trademarked terms are included in this manual.

Diffusion is trademark of Push Technology Ltd.

ActionScript, Adobe, Flash, and Flexare registered trademarks of Adobe Systems Incorporated.

AIX™, Bluemix®, Cast Iron®, and WebSphere® are trademarks of IBM.

Amazon and Amazon EC2 are trademarks of Amazon.

Android and Chrome are trademarks of Google Inc.

Ant, Apache, Apache Derby™, Apache Tomcat™, and Maven are trademarks of The Apache Software Foundation.

BlackBerry® is a registered trademark of RIM.

CentOS and Red Hat are trademarks or registered trademarks of Red Hat, Inc.

Dell™ is trademark of Dell, Inc.

Docker is trademarks or registered trademarks of Docker, Inc. in the United States and/or other countries.

Eclipse is a trademark of the Eclipse Foundation, Inc.

F5 is a registered trademark of F5 Networks, Inc.

Firefox is a registered trademark of Mozilla Foundation.

Hazelcast is a trademark of Hazelcast Inc.

Intel and Xeon are trademarks of Intel Corporation.

Internet Explorer, Microsoft, Silverlight, and Windows are trademarks or registered trademarks of Microsoft Corporation.

iOS is a registered trademark of Cisco.

Java, JavaScript, Oracle, and Solaris™ are trademarks or registered trademarks of Oracle Corporation.

Joyent is a trademarks of Joyent.

Linux is a trademark of Linus Torvalds.

Mac®, OS X, Safari, and Siri® are registered trademarks of Apple Inc.

Nagios® is a registered trademark of Nagios Enterprises.

Opera is a registered trademark of Opera Software ASA.

Splunk is a trademark of Splunk, Inc.

SUSE® is a registered trademark of SUSE LLC.

TIBCO Enterprise Message Service is a trademark of TIBCO Software Inc.

Ubuntu is a registered trademark of Canonical Ltd.

UNIX is a registered trademark of The Open Group.

VeriSign® is a registered trademark of VeriSign, Inc.

VMware® and VMware vSphere are registered trademarks of VMware, Inc.

Appendix D

Copyright Notices

Diffusion uses third party, open source software. The rights to this software are not owned by Push Technology and the software is distributed under different licensing agreements. The distribution and use of third-party software is subject to the applicable terms.

The following sections list the software used, their licenses, copyright notices and disclaimers.

In this section:

- ANTLR
- Apache Commons Codec
- Apache Portable Runtime
- Bootstrap
- CocoaAsyncSocket
- concurrent-trees
- CQEngine
- cron4j
- d3
- FastColoredTextBox
- Fluent validation
- Fluidbox
- GeoIP API
- GeoLite City Database
- geronimo-jms_1.1_spec
- Google code prettify
- hashmap
- Hazelcast
- HPPC
- htmlcompressor
- inherits
- JCIP Annotations
- JCTools

- jQuery
- json-simple
- JZlib
- Knockout
- loglevel
- long
- Metrics
- Minimal JSON
- NLog
- opencsv
- OpenSSL
- PCRE
- Picocontainer
- Protocol Buffers
- Rickshaw
- Servlet API
- SLF4J
- SocketRocket
- Tabber
- Tapestry (Plastic)
- TrueLicense
- when
- Licenses

ANTLR

Version 4.3

http://www.antlr.org

ANTLR is distributed under the BSD 3-clause License.

Copyright (c) 2014 Terence Parr, Sam Harwell

Apache Commons Codec

Version 1.8

http://commons.apache.org/codec/

Apache Commons Codec is distributed under the Apache License 2.0.

Copyright 2002-2011 The Apache Software Foundation. All Rights Reserved

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
Apache Commons Codec
Copyright 2002-2011 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

---------------------------------------------------------------------------
src/test/org/apache/commons/codec/language/
DoubleMetaphoneTest.java contains
test data from http://aspell.sourceforge.net/test/batch0.tab.

Copyright (C) 2002 Kevin Atkinson (kevina@gnu.org). Verbatim
 copying
and distribution of this entire article is permitted in any
 medium,
provided this notice is preserved.
---------------------------------------------------------------------------
```

Apache Portable Runtime

Version 8.3.3

http://apr.apache.org

APR is distributed under the Apache 2.0 License.

Copyright (c) 2015 The Apache Software Foundation

Bootstrap

Version: 3.2.0

https://github.com/twbs/bootstrap/

Bootstrap is distributed under the MIT License.

Copyright (c) 2011-2014 Twitter, Inc

Additional notes

We also use Glyphicons, which are included as part of Bootstrap.

CocoaAsyncSocket

Version 7.3.4

https://github.com/robbiehanson/CocoaAsyncSocket

The CocoaAsyncSocket project is in the public domain.

The original TCP version (AsyncSocket) was created by Dustin Voss in January 2003.

Updated and maintained by Deusty LLC and the Apple development community.

concurrent-trees

Version 2.4.0

https://code.google.com/p/concurrent-trees/

concurrent-trees is distributed under the Apache 2.0 License.

Copyright 2012-2013 Niall Gallagher

CQEngine

Version 2.0.3

https://code.google.com/p/cqengine/

CQEngine is distributed under the Apache 2.0 License.

Copyright 2012-2015 Niall Gallagher

cron4j

Version 2.2.5

http://www.sauronsoftware.it/projects/cron4j/

Cron4j is distributed under the LGPL 2.1.

Copyright (C) 2007-2010 Carlo Pelliccia (www.sauronsoftware.it)

Source code is available from the following location: http://sourceforge.net/projects/cron4j/files/cron4j/2.2.5/cron4j-2.2.5.zip/download/

For a fee, Push Technology can also provide this source on a CD. To request a copy, contact support@pushtechnology.com.

d3

Version 3.2.1

http://d3js.org/

d3 is distributed under the Apache License 2.0.

Copyright (c) 2010-2014, Michael Bostock

FastColoredTextBox

https://github.com/PavelTorgashov/FastColoredTextBox

FastColoredTextBox distributed under the LGPL 3.0 or later.

Copyright (C) Pavel Torgashov, 2011-2014.

Source code is available at the following location: https://github.com/PavelTorgashov/FastColoredTextBox

For a fee, Push Technology can also provide this source on a CD. To request a copy, contact support@pushtechnology.com.

Fluent validation

Version 3.3.1.0

http://fluentvalidation.codeplex.com/

Fluent validation is distributed under the Apache License 2.0.

Copyright Jeremy Skinner

Fluidbox

https://github.com/terrymun/Fluidbox

Fluidbox is distributed under the MIT License.

Copyright (c) 2014 Terry Mun

GeoIP API

Version 1.2.13

http://www.maxmind.com/en/opensource

The GeoIP API is distributed under the LGPL 2.1 or later.

Copyright (C) 2003 MaxMind LLC. All Rights Reserved

GeoLite City Database

http://www.maxmind.com/en/opensource

The GeoLite City Database is distributed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.

Copyright MaxMind LLC

geronimo-jms_1.1_spec

Version 1.1

http://geronimo.apache.org/

geronimo-jms_1.1_spec is distributed under the Apache License 2.0

Copyright 2003-2006 The Apache Software Foundation

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
Apache Geronimo
Copyright 2003-2006 The Apache Software Foundation

This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).
```

Google code prettify

https://code.google.com/p/google-code-prettify/

Prettify is distributed under the Apache 2.0 License.

Copyright (C) 2006 Google Inc.

hashmap

Version: 1.1.0

https://github.com/flesler/hashmap

hashmap is distributed under the MIT License.

Copyright (c) 2012-2013 Ariel Flesler aflesler@gmail.com

Hazelcast

Version 3.5

http://www.hazelcast.org/

Hazelcast is distributed under the Apache License 2.0

Copyright (c) 2008-2015, Hazelcast, Inc. All Rights Reserved.

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
**
**  NOTICE file corresponding to the section 4 (d) of the Apache
 License,
**  Version 2.0, in this case for the Hazelcast distribution.
**

The end-user documentation included with a redistribution, if any,
must include the following acknowledgement:

 "This product includes software developed by the Hazelcast
  Project (http://www.hazelcast.com)."

Alternately, this acknowledgement may appear in the software
 itself,
if and wherever such third-party acknowledgements normally appear.

The name "Hazelcast" must not be used to endorse or promote
 products
derived from this software without prior written permission. For
written permission, please contact talip@hazelcast.com or
 fuad@hazelcast.com
```

```
Copyright (c) 2008-2014 Hazelcast Inc.  All rights reserved.
```

HPPC

Version 0.6.0

http://labs.carrotsearch.com/hppc.html

HPPC is distributed under the Apache License 2.0.

Copyright 2010-2013, Carrot Search s.c., Boznicza 11/56, Poznan, Poland

htmlcompressor

Version 1.5.2

http://code.google.com/p/htmlcompressor/

The htmlcompressor is distributed under the Apache License 2.0.

Copyright 2009-2011 Sergiy Kovalchuk

Additional notices: Apache License 2.0 Notice

inherits

Version 2.0.1

https://github.com/isaacs/inherits

Inherits is distributed under the ISC License.

Copyright (c) Isaac Z. Schlueter.

JCIP Annotations

Version 1

https://github.com/stephenc/jcip-annotations

jcip-annotations is distributed under the Apache License 2.0

Copyright 2013 Stephen Connolly.

JCTools

Version 1.0

https://github.com/JCTools/JCTools

JCTools is distributed under the Apache License 2.0

Copyright 2015 Nitsan Wakart.

jQuery

Version: 1.7.1

https://jquery.org/

jQuery is distributed under the MIT License.

Copyright 2014 jQuery Foundation and other contributors http://jquery.com/

json-simple

Version 1.1.1

https://code.google.com/p/json-simple/

json-simple is distributed under the Apache License 2.0.

Copyright (c) Yidong Fang, Chris Nokleberg

JZlib

Version 1.02

http://www.jcraft.com/jzlib/

JZlib is distributed under the BSD 3-clause License. This has not been modified, it has been compiled from the source code for distribution.

Copyright 2000-2011 ymnk, JCraft,Inc. All rights reserved.

Knockout

Version 2.1.0

http://knockoutjs.com/

Knockout is distributed under the MIT License.

Copyright (c) Steven Sanderson, the Knockout.js team, and other contributors

loglevel

Version: 1.1.0

https://github.com/pimterry/loglevel

loglevel is distributed under the MIT License.

Copyright (c) 2013 Tim Perry

long

Version: 2.1.0

https://github.com/dcodeIO/Long.js

long is distributed under the Apache License 2.0.

Copyright 2013 Daniel Wirtz dcode@dcode.io

Copyright 2009 The Closure Library Authors. All Rights Reserved.

Metrics

Version 3.0.0-BETA

http://metrics.codahale.com/

Metrics is distributed under the Apache License 2.0.

Copyright (c) 2010-2013 Coda Hale, Yammer.com

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
Metrics
Copyright 2010-2013 Coda Hale and Yammer, Inc.

This product includes software developed by Coda Hale and Yammer,
 Inc.

This product includes code derived from the JSR-166 project
 (ThreadLocalRandom, Striped64,
LongAdder), which was released with the following comments:

    Written by Doug Lea with assistance from members of JCP
 JSR-166
    Expert Group and released to the public domain, as explained
 at
    http://creativecommons.org/publicdomain/zero/1.0/
```

Minimal JSON

https://github.com/ralfstx/minimal-json

Minimal JSON is distributed under the MIT License.

Copyright (c) 2014, 2015 EclipseSource

NLog

Version 3.1.0

https://github.com/NLog/NLog/

NLog is distributed under the BSD 3-clause License.

Copyright (c) 2004-2011 Jaroslaw Kowalski <jaak@jkowalski.net>

opencsv

Version 2.3

http://opencsv.sourceforge.net/

opencsv is distributed under the Apache License 2.0.

Copyright 2005 Bytecode Pty Ltd.

OpenSSL

Version 1.0.2a

https://www.openssl.org/

OpenSSL is distributed under the OpenSSL and SSLeay Licenses.

PCRE

Version 1.5.2

http://www.pcre.org/

PCRE is distributed under the BSD 3-clause License.

THE BASIC LIBRARY FUNCTIONS

Written by: Philip Hazel

Email local part: ph10

Email domain: cam.ac.uk

University of Cambridge Computing Service,

Cambridge, England.

Copyright (c) 1997-2015 University of Cambridge

All rights reserved.

PCRE2 JUST-IN-TIME COMPILATION SUPPORT

Written by: Zoltan Herczeg

Email local part: hzmester

Emain domain: freemail.hu

Copyright(c) 2010-2015 Zoltan Herczeg

All rights reserved.

STACK-LESS JUST-IN-TIME COMPILER

Written by: Zoltan Herczeg

Email local part: hzmester

Emain domain: freemail.hu

Copyright(c) 2009-2015 Zoltan Herczeg

All rights reserved.

Picocontainer

Version 2.14.3

http://picocontainer.codehaus.org/

Picocontainer is distributed under the BSD 3-clause License.

Copyright (c) 2003-2008 PicoContainer Organization. All rights reserved.

Protocol Buffers

Version 2.6.1

http://code.google.com/p/protobuf/

Google's Protocol Buffers are distributed under the BSD 3-clause License.

Copyright 2008 Google Inc. All rights reserved.

Rickshaw

http://code.shutterstock.com/rickshaw/

Rickshaw is distributed under the MIT License.

Copyright (C) 2011-2013 by Shutterstock Images, LLC

Servlet API

http://jetty.mortbay.org/project/modules/servlet-api-2.5

Servlet API is distributed under the CDDL v1.0 License.

SLF4J

Version 1.7.7

http://www.slf4j.org/

SLF4J is distributed under the MIT License.

Copyright (c) 2004-2013 QOS.ch All rights reserved.

SocketRocket

Version 0.3.1-beta2

https://github.com/square/SocketRocket

SocketRocket is distributed under the Apache License 2.0.

Copyright 2012 Square Inc.

Tabber

Version: 1.9

http://www.barelyfitz.com/projects/tabber/

Tabber is distributed under the MIT License.

Copyright (c) 2006 Patrick Fitzgerald pat@barelyfitz.com

Tapestry (Plastic)

Version 5.3.7

http://tapestry.apache.org/

Tapestry is distributed under the Apache License 2.0.

Copyright 2011, 2012 The Apache Software Foundation

Additional notices

The following information is included in the `NOTICE.txt` file that accompanies the source:

```
This product includes software developed by
The Apache Software Foundation (http://www.apache.org/).

Please refer to the NOTICE.txt in each sub-module to
identify further dependencies.

The Maven central repository is the preferred method to download
 Tapestry
and its dependencies. The binary archive includes just basic
dependencies for tapestry-core; using other modules (such as
tapestry-hibernate or any of the others) requires downloading
additional dependencies. Please refer to the Maven POM for each
 module
to identify its dependencies.
```

TrueLicense

Version 1.33

http://truelicense.java.net/

This version of TrueLicense was distributed under the EPL.

Copyright 2005-2012 Schlichtherle IT Services

when

Version 3.7.2

https://github.com/cujojs/when

http://cujojs.com/

When is distributed under the MIT License.

Copyright (c) 2011 Brian Cavalier

Licenses

The following licenses are used by the third party, open source software that is distributed with Diffusion.

Apache License 2.0

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within

Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Related Links

Apache Commons Codec on page 814

Apache Portable Runtime on page 814

CQEngine on page 815

concurrent-trees on page 815

d3 on page 815

Fluent validation on page 816

geronimo-jms_1.1_spec on page 816

Hazelcast on page 817

HPPC on page 818

htmlcompressor on page 818
JCIP Annotations on page 818
JCTools on page 818
json-simple on page 819
long on page 819
Metrics on page 819
Google code prettify on page 817
opencsv on page 820
SocketRocket on page 822
Tapestry (Plastic) on page 822

BSD 3-clause License

Copyright (c) <YEAR>, <OWNER>

All rights reserved.

Note: The copyright statement above is included in its completed form in the sections of this document specific to the individual products covered by this license.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JCRAFT, INC. OR ANY CONTRIBUTORS TO THIS SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Related Links

ANTLR on page 814
Protocol Buffers on page 821
JZlib on page 819
NLog on page 820
PCRE on page 820
Picocontainer on page 821

Common Development and Distribution License

Sun Microsystems, Inc. ("Sun") ENTITLEMENT for SOFTWARE

Licensee/Company: Entity receiving Software.

Effective Date: Date of delivery of the Software to You.

Software: JavaMail 1.4.

License Term: Perpetual (subject to termination under the SLA).

Licensed Unit: Software Copy.

Licensed unit Count: Unlimited.

Permitted Uses:

1. You may reproduce and use the Software for Individual, Commercial, or Research and Instructional Use for the purposes of designing, developing, testing, and running Your applets and application("Programs").

2. Subject to the terms and conditions of this Agreement and restrictions and exceptions set forth in the Software's documentation, You may reproduce and distribute portions of Software identified as a redistributable in the documentation ("Redistributable"), provided that:

(a) you distribute Redistributable complete and unmodified and only bundled as part of Your Programs,

(b) your Programs add significant and primary functionality to the Redistributable,

(c) you distribute Redistributable for the sole purpose of running your Programs,

(d) you do not distribute additional software intended to replace any component(s) of the Redistributable,

(e) you do not remove or alter any proprietary legends or notices contained in or on the Redistributable.

(f) you only distribute the Redistributable subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and

(g) you agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Redistributable.

3. Java Technology Restrictions. You may not create, modify, or change the behavior of, or authorize your licensees to create, modify, or change the behavior of, classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

B. Sun Microsystems, Inc. ("Sun") SOFTWARE LICENSE AGREEMENT

READ THE TERMS OF THIS AGREEMENT ("AGREEMENT") CAREFULLY BEFORE OPENING SOFTWARE MEDIA PACKAGE. BY OPENING SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" (OR "EXIT") BUTTON AT THE END OF THIS AGREEMENT. IF YOU HAVE SEPARATELY AGREED TO LICENSE TERMS ("MASTER

TERMS") FOR YOUR LICENSE TO THIS SOFTWARE, THEN SECTIONS 1-5 OF THIS AGREEMENT ("SUPPLEMENTAL LICENSE TERMS") SHALL SUPPLEMENT AND SUPERSEDE THE MASTER TERMS IN RELATION TO THIS SOFTWARE.

1. Definitions.

(a) "Entitlement" means the collective set of applicable documents authorized by Sun evidencing your obligation to pay associated fees (if any) for the license, associated Services, and the authorized scope of use of Software under this Agreement.

(b) "Licensed Unit" means the unit of measure by which your use of Software and/or Service is licensed, as described in your Entitlement.

(c) "Permitted Use" means the licensed Software use(s) authorized in this Agreement as specified in your Entitlement. The Permitted Use for any bundled Sun software not specified in your Entitlement will be evaluation use as provided in Section 3.

(d) "Service" means the service(s) that Sun or its delegate will provide, if any, as selected in your Entitlement and as further described in the applicable service listings at www.sun.com/service/servicelist.

(e) "Software" means the Sun software described in your Entitlement. Also, certain software may be included for evaluation use under Section 3.

(f) "You" and "Your" means the individual or legal entity specified in the Entitlement, or for evaluation purposes, the entity performing the evaluation.

2. License Grant and Entitlement.

Subject to the terms of your Entitlement, Sun grants you a nonexclusive, nontransferable limited license to use Software for its Permitted Use for the license term. Your Entitlement will specify (a) Software licensed, (b) the Permitted Use, (c) the license term, and (d) the Licensed Units.

Additionally, if your Entitlement includes Services, then it will also specify the (e) Service and (f) service term.

If your rights to Software or Services are limited in duration and the date such rights begin is other than the purchase date, your Entitlement will provide that beginning date(s).

The Entitlement may be delivered to you in various ways depending on the manner in which you obtain Software and Services, for example, the Entitlement may be provided in your receipt, invoice or your contract with Sun or authorized Sun reseller. It may also be in electronic format if you download Software.

3. Permitted Use.

As selected in your Entitlement, one or more of the following Permitted Uses will apply to your use of Software. Unless you have an Entitlement that expressly permits it, you may not use Software for any of the other Permitted Uses. If you don't have an Entitlement, or if your Entitlement doesn't cover additional software delivered to you, then such software is for your Evaluation Use.

(a) Evaluation Use. You may evaluate Software internally for a period of 90 days from your first use.

(b) Research and Instructional Use. You may use Software internally to design, develop and test, and also to provide instruction on such uses.

(c) Individual Use. You may use Software internally for personal, individual use.

(d) Commercial Use. You may use Software internally for your own commercial purposes.

(e) Service Provider Use. You may make Software functionality accessible (but not by providing Software itself or through outsourcing services) to your end users in an extranet deployment, but not to your affiliated companies or to government agencies.

4. Licensed Units.

Your Permitted Use is limited to the number of Licensed Units stated in your Entitlement. If you require additional Licensed Units, you will need additional Entitlement(s).

5. Restrictions.

(a) The copies of Software provided to you under this Agreement are licensed, not sold, to you by Sun. Sun reserves all rights not expressly granted. (b) You may make a single archival copy of Software, but otherwise may not copy, modify, or distribute Software. However if the Sun documentation accompanying Software lists specific portions of Software, such as header files, class libraries, reference source code, and/or redistributable files, that may be handled differently, you may do so only as provided in the Sun documentation. (c) You may not rent, lease, lend or encumber Software. (d) Unless enforcement is prohibited by applicable law, you may not decompile, or reverse engineer Software. (e) The terms and conditions of this Agreement will apply to any Software updates, provided to you at Sun's discretion, that replace and/or supplement the original Software, unless such update contains a separate license. (f) You may not publish or provide the results of any benchmark or comparison tests run on Software to any third party without the prior written consent of Sun. (g) Software is confidential and copyrighted. (h) Unless otherwise specified, if Software is delivered with embedded or bundled software that enables functionality of Software, you may not use such software on a stand-alone basis or use any portion of such software to interoperate with any program(s) other than Software. (i) Software may contain programs that perform automated collection of system data and/or automated software updating services. System data collected through such programs may be used by Sun, its subcontractors, and its service delivery partners for the purpose of providing you with remote system services and/or improving Sun's software and systems. (j) Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility and Sun and its licensors disclaim any express or implied warranty of fitness for such uses. (k) No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

6. Term and Termination.

The license and service term are set forth in your Entitlement(s). Your rights under this Agreement will terminate immediately without notice from Sun if you materially breach it or take any action in derogation of Sun's and/or its licensors' rights to Software. Sun may terminate this Agreement should any Software become, or in Sun's reasonable opinion likely to become, the subject of a claim of intellectual property infringement or trade secret misappropriation. Upon termination, you will cease use of, and destroy, Software and confirm compliance in writing to Sun. Sections 1, 5, 6, 7, and 9-15 will survive termination of the Agreement.

7. Java Compatibility and Open Source.

Software may contain Java technology. You may not create additional classes to, or modifications of, the Java technology, except under compatibility requirements available under a separate agreement available at www.java.net.

Sun supports and benefits from the global community of open source developers, and thanks the community for its important contributions and open standards-based technology, which Sun has adopted into many of its products.

Please note that portions of Software may be provided with notices and open source licenses from such communities and third parties that govern the use of those portions, and any licenses granted hereunder do not alter any rights and obligations you may have under such open source licenses, however, the disclaimer of warranty and limitation of liability provisions in this Agreement will apply to all Software in this distribution.

8. Limited Warranty.

Sun warrants to you that for a period of 90 days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software. Some states do not allow limitations on certain implied warranties, so the above may not apply to you. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

9. Disclaimer of Warranty.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

10. Limitation of Liability.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose. Some states do not allow the exclusion of incidental or consequential damages, so some of the terms above may not be applicable to you.

11. Export Regulations.

All Software, documents, technical data, and any other materials delivered under this Agreement are subject to U.S. export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with these laws and regulations and acknowledge that you have the responsibility to obtain any licenses to export, re-export, or import as may be required after delivery to you.

12. U.S. Government Restricted Rights.

If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

13. Governing Law.

Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

14. Severability.

If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

15. Integration.

This Agreement, including any terms contained in your Entitlement, is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

Please contact Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054 if you have questions.

Related Links

Servlet API on page 822

Eclipse Public License – v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

- a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
- b) in the case of each subsequent Contributor:
  - i) changes to the Program, and
  - ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program. "Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
  - i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
  - ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
  - iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
  - iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other

Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this

Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Related Links

TrueLicense on page 823

ISC License –

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Related Links

inherits on page 818

The GNU Lesser General Public License, version 2.1 (LGPL-2.1)

GNU Lesser General Public License

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages – typically libraries – of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license

practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Related Links

cron4j on page 815

GeoIP API on page 816

The GNU Lesser General Public License, version 3.0 (LGPL-3.0)

This license is a set of additional permissions added to version 3 of the GNU General Public License.

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Related Links

FastColoredTextBox on page 816

The MIT License (MIT)

Copyright (c) <year> <copyright holders>

Note: The copyright statement above is included in its completed form in the sections of this document specific to the individual products covered by this license.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Related Links

Bootstrap on page 814

Fluidbox on page 816

hashmap on page 817

jQuery on page 818

Knockout on page 819

loglevel on page 819

Rickshaw on page 821

SLF4J on page 822

Tabber on page 822

Minimal JSON on page 820

when on page 823

OpenSSL and SSLeay Licenses

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts.

OpenSSL License

Copyright (c) 1998-2011 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)

All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

   "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

   The word 'cryptographic' can be left out if the rouines from the library being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Related Links

OpenSSL on page 820